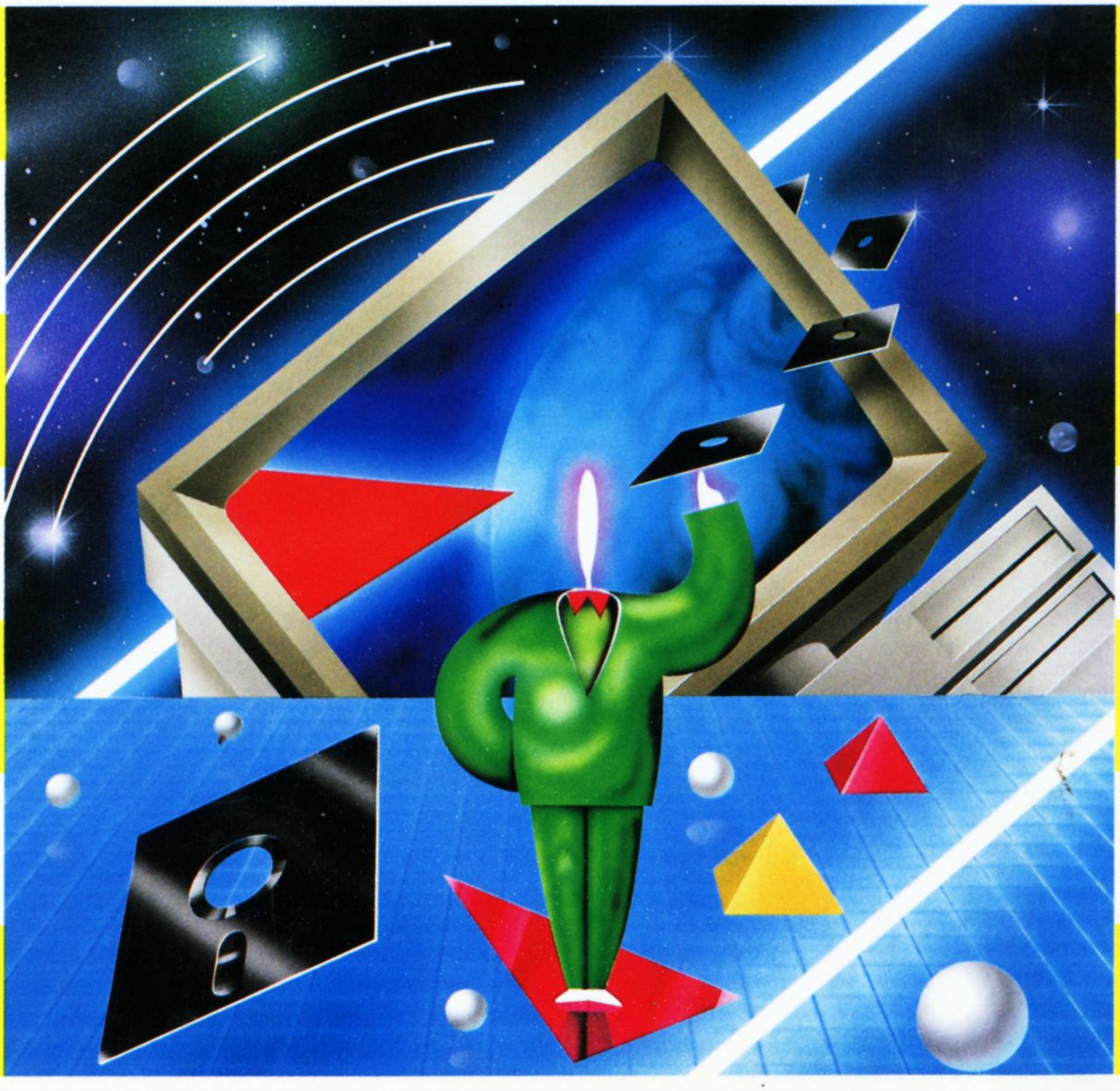
絵でわかる

はじめてのPC-98

基礎知識と操作&BASIC

田中一郎



新星出版社

絵でわかる

はじめてのPC-98

基礎知識と操作＆BASIC

田中一郎

新星出版社

はじめに

パソコンの普及によって、日本中いたるところコンピュータ化が進みました。それに即応して高校はもちろんのこと、中学、小学校においてさえも、コンピュータが必修科目になろうとしています。このようになると誰もが常識として、パソコンに関して一通りのことは知っておかなければならなくなったといえましょう。

ところで、初心者を対象とするパソコンの入門書をみてみると、キーボードの操作の仕方と、パソコンのプログラミング言語（プムログラムを作る言葉）であるベーシックの解説がほとんどです。これでは途中で壁にぶつかってわからなくなったとき、その壁を乗り越えることができずに、途中で放棄してしまうことになりかねません。

私の経験からいうと、初めてパソコンに接する人がもっとも大切なことは、パソコンの基本的な構造を知ることです。そのうえでベーシックを習得すると、壁にぶつかってわからなくなっても、何とか解決する方法を見つけ出すことができて、途中で放棄しないですみます。

この本はこのような考えにもとづいて、パソコンの基本的な構造を誰にでもわかるようにやさしく具体的に説明し、そのあとキーボードの操作、ベーシックの働きや使い方などを説明しています。

この本が、初めてパソコンに接する人、またパソコンに接して間もない人など、すべての人のお役に立てばと思っています。

田中　一郎

もくじ

第1章 16ビット・パソコンの特徴と利用

第2章 16ビット・パソコンの仕組み

第3章 ■ パソコン操作の基礎知識

第4章 BASICによるプログラミング

第5章　ディスクの操作

第6章　日本語の入力

第1章

16ビット・パソコンの特徴と利用



16ビット・パソコン（PC-9801VX21）

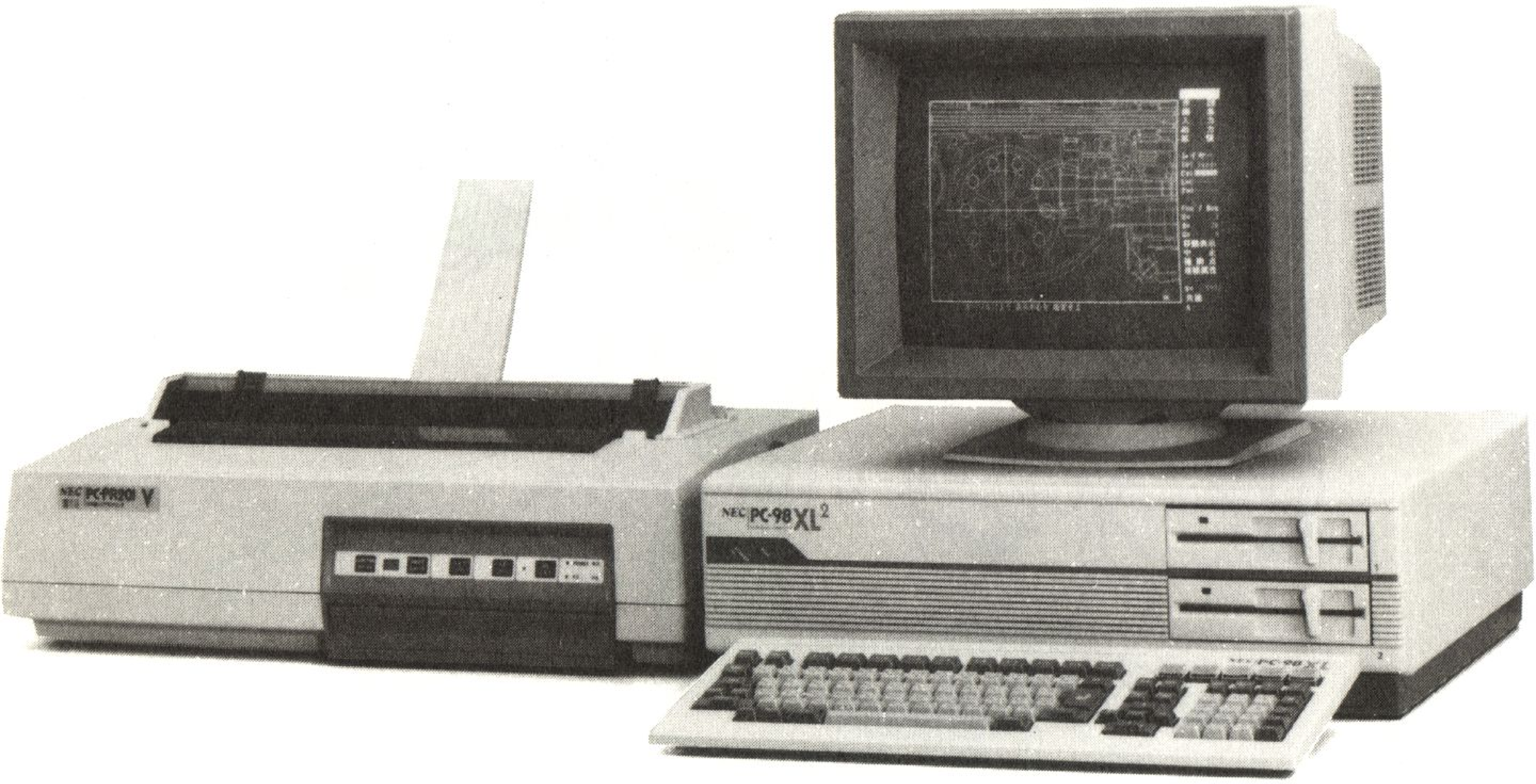
16ビットパソコンの特徴

パーソナルコンピュータ、つまりパソコンは、8ビット・パソコンと16ビット・パソコンの二種類が主流ですが、最近では32ビット・パソコンも登場しています。

周知のように、最初にパソコンとして一般に普及したのは、8ビット・パソコンです。

もちろん、8ビット・パソコンが一般に普及したといっても、パソコンの機能やアプリケーション・ソフトが限られていたので、主として個人の趣味、つまりゲームやパソコンの学習といった、いわゆる「ホビー・ユース」であって、ビジネスなどに広く利用されるというものではありませんでした。

このあと、NECのPC－9801シリーズ、



32ビット・パソコン(PC-98XL²)

富士通のFM－Rシリーズ、シャープのX－68000、また日立のB16シリーズなど、高性能でしかも手頃な価格の16ビット・パソコンが登場し、優れたアプリケーション・ソフトの開発とあいまって、はじめてビジネスなどに広く利用されるようになったわけです。つまり、16ビット・パソコンの登場によってはじめて、本当の意味でパソコンが一般に普及したといえます。

ところで、8ビット・パソコンと比較した場合の16ビット・パソコンの特徴としては、いろいろ取りあげることができますが、なかでも、

処理スピードが速い

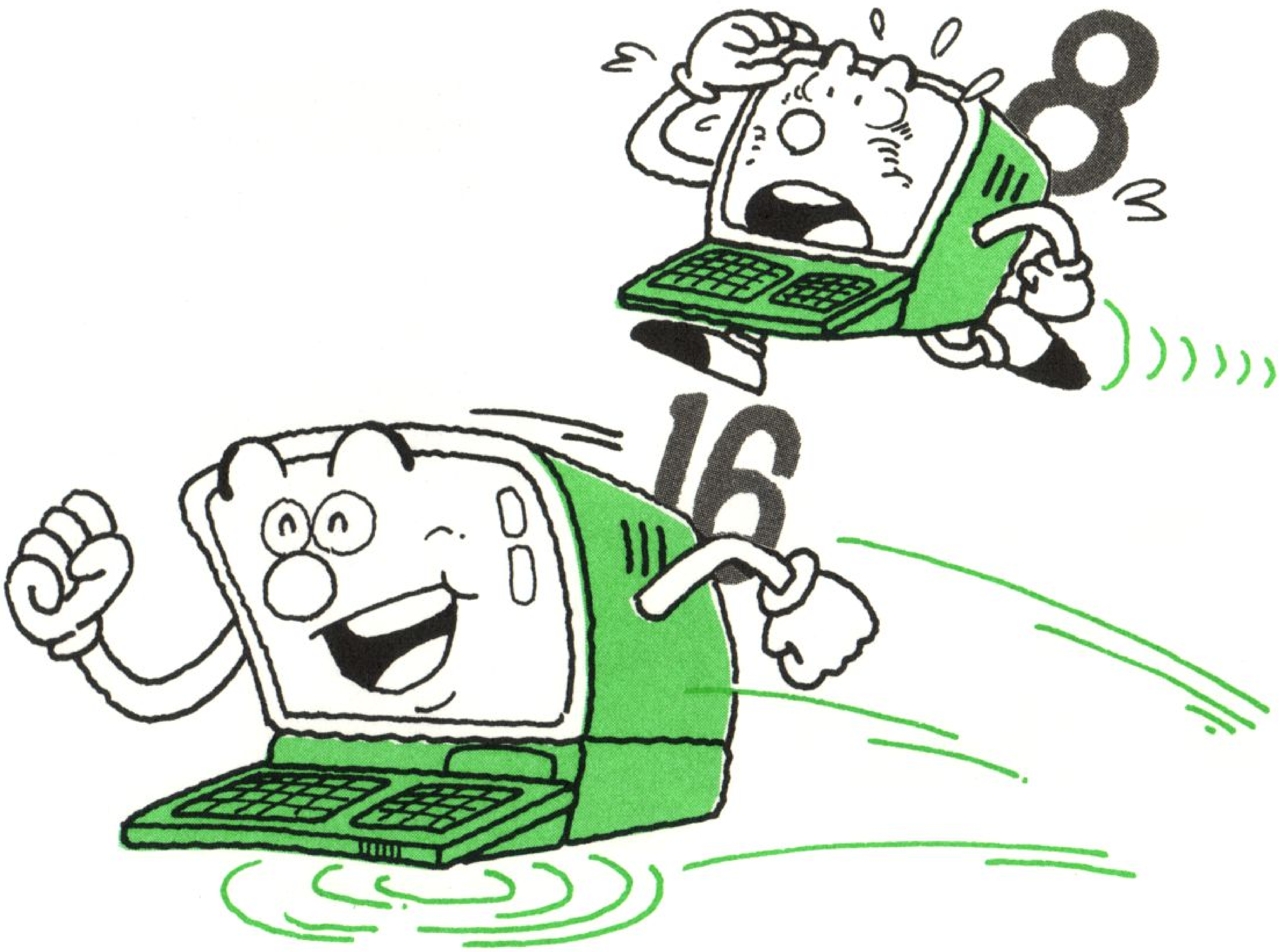
記憶回路(メモリ)の記憶容量が大きい

という二つをあげることができます。

一度に16ビット単位で処理

処理スピードというのは、コンピュータが与えられた仕事を処理する速さのことです。

8ビット・パソコンの"8ビット"、16ビット・



16ビット・パソコンのほうが処理スピードが速い

パソコンの〝16ビット〟というのは、一度に処理することができる仕事の単位のことです。

16ビットは8ビットの2倍ですから、16ビット・パソコンは8ビット・パソコンの2倍の仕事を一度に処理することができる、つまり、2倍の速さの処理スピードを持っているということです。

しかし、実際には、16ビット・パソコンといえども8ビット・パソコンの2倍の速さの処理スピードで仕事を処理することはできません。

というのは、計算やグラフィックなど、処理した結果をディスプレイに表示したり、プリンタから打ち出したりするなど、時間のかかる仕事があるからです。

とはいっても、8ビット・パソコンに比べれば16ビット・パソコンのほうが数段速い処理スピードで、与えられた仕事を処理していきます。

記憶容量が大きい

コンピュータは、与えられた仕事をする場合、



16ビット・パソコンは、はるかに大きい記憶容量をもっている

その仕事をするのに必要な情報を一度記憶場所に記憶してから行いますが、記憶容量とは、この情報を記憶しておくことができる記憶場所の大きさのことです。

このため、コンピュータは、記憶容量が小さいと少量の情報しか記憶することができないので、小さい仕事しか行うことができませんが、反対に大きな記憶容量を持っていれば、多量の情報を記憶することができるので、大きな仕事ができるようになるのです。

コンピュータとして、どちらが優れているかはいうまでもないでしょう。

ところで、８ビット・パソコンと16ビット・パソコンの記憶容量はどのくらいかというと、８ビット・パソコンの記憶容量は、最大で64ＫＢ（キロバイト）ですが、16ビット・パソコンの記憶容量は、最低で256ＫＢ（キロバイト）というように、16ビット・パソコンは８ビット・パソコンに比べてはるかに大きな記憶容量を持っています（キロバイトについては44頁参照）。

MS-DOSとアプリケーション・ソフトの利用

与えられた仕事の処理スピードが速い、また、非常に大きな記憶容量を持っているという、ただそれだけの理由で、16ビット・パソコンがビジネスなどに広く利用されるようになったわけではありません。

処理スピードが速い、非常に大きな記憶容量を持っているということを利用して、大型コンピュータのOS（オペレーティング・システム）とおなじように、16ビット・パソコン用のOSが開発されたこと、また、そのOSの機能を利用した、いろいろな優れたアプリケーション・ソフトが開発され、市販されるようになったからです。

16ビット・パソコン用のOSのことを、DOS（ディスク・オペレーティング・システム）といいます。

DOSには、MS-DOSとCP／M86がありますが、多くの16ビット・パソコンに使われているのは、このうちのMS-DOSです。

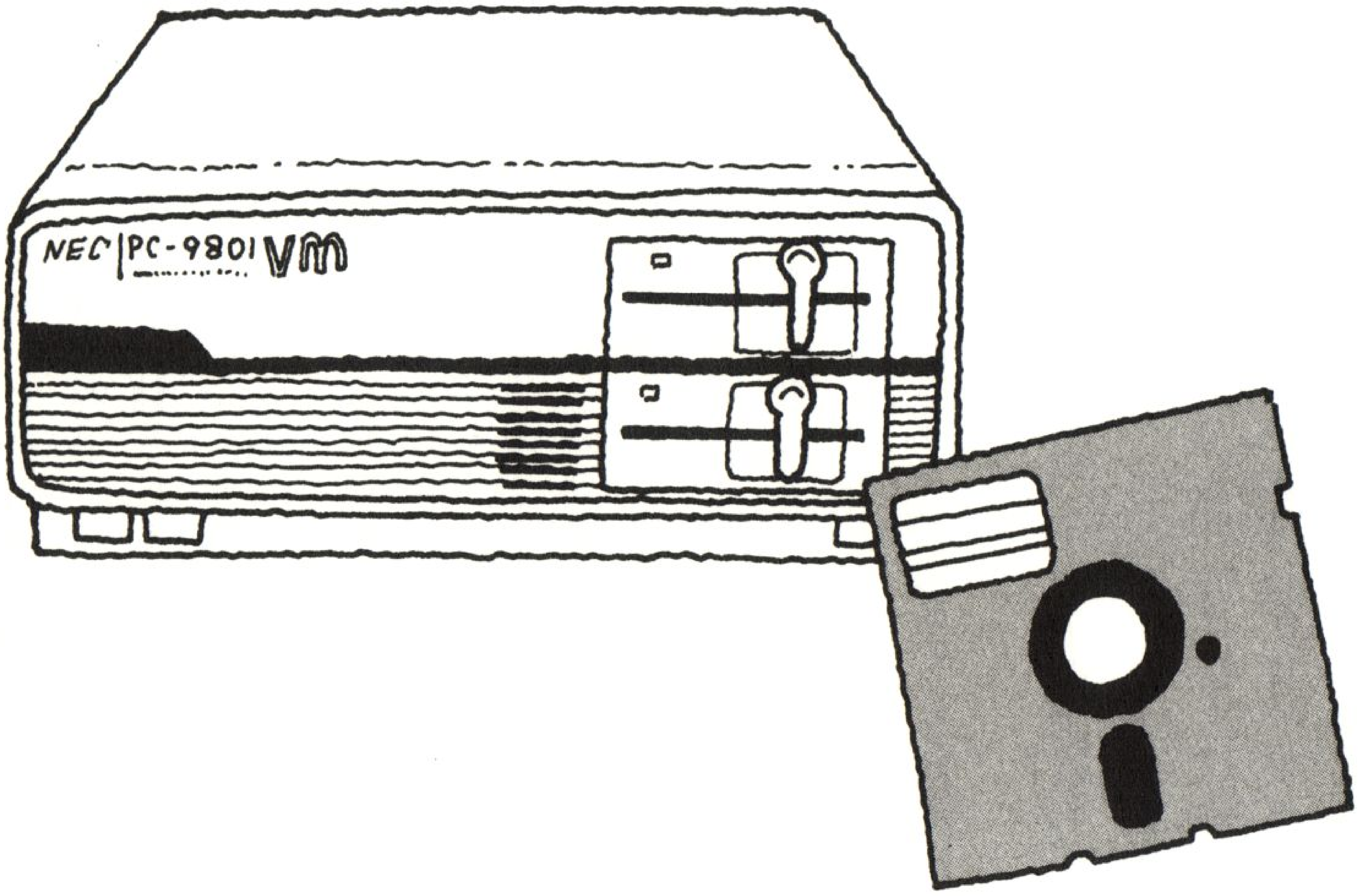
このDOSの機能を利用した、いろいろなプログラムのことをアプリケーション・ソフト、あるいはアプリケーション・プログラムといいます。

アプリケーション・ソフトとは、一つの仕事を行う目的で作られたプログラムで、よく知られているソフトとしては、ワープロソフト（新・一太郎）、表計算ソフト（マルチプラン）、図形プロセッサ（花子）などがあります。

これらについての詳しいことは入門書に譲り、ここでは、簡単に紹介しておくことにします。

MS-DOS

MS-DOSの〝MS〟は、DOSを開発した会社の名前の略、〝DOS〟は、ディスク・オペレーティング・システムの略です。



パソコンのディスク装置とディスク

このうちの〝ディスク〟とは、16ビット・パソコンのボディに組み込まれているディスク装置と、そのディスク装置に挿入してデータを記録するディスクのことです。〝オペレーティング〟は「操作する」という意味、〝システム〟とは「操作するためのいろいろなプログラムの集り」という意味ですから、DOSとは、ディスク装置とディスクを操作するためのソフトウェア（プログラム）ということになります。

では、MS-DOSのディスク装置とディスクを操作するということについて説明します。

キーボードのキーを叩いてパソコン本体内部に入力したデータは、ディスク装置に挿入したディスクに記録して保管します。

データをディスクに記録する場合、データをファイルという形で記録しますから、ディスクにデータを記録することを、**ファイルへの書き込み**といいます。

また、ディスクに記録したデータを必要とするときは、ディスク装置にディスクを挿入して、記

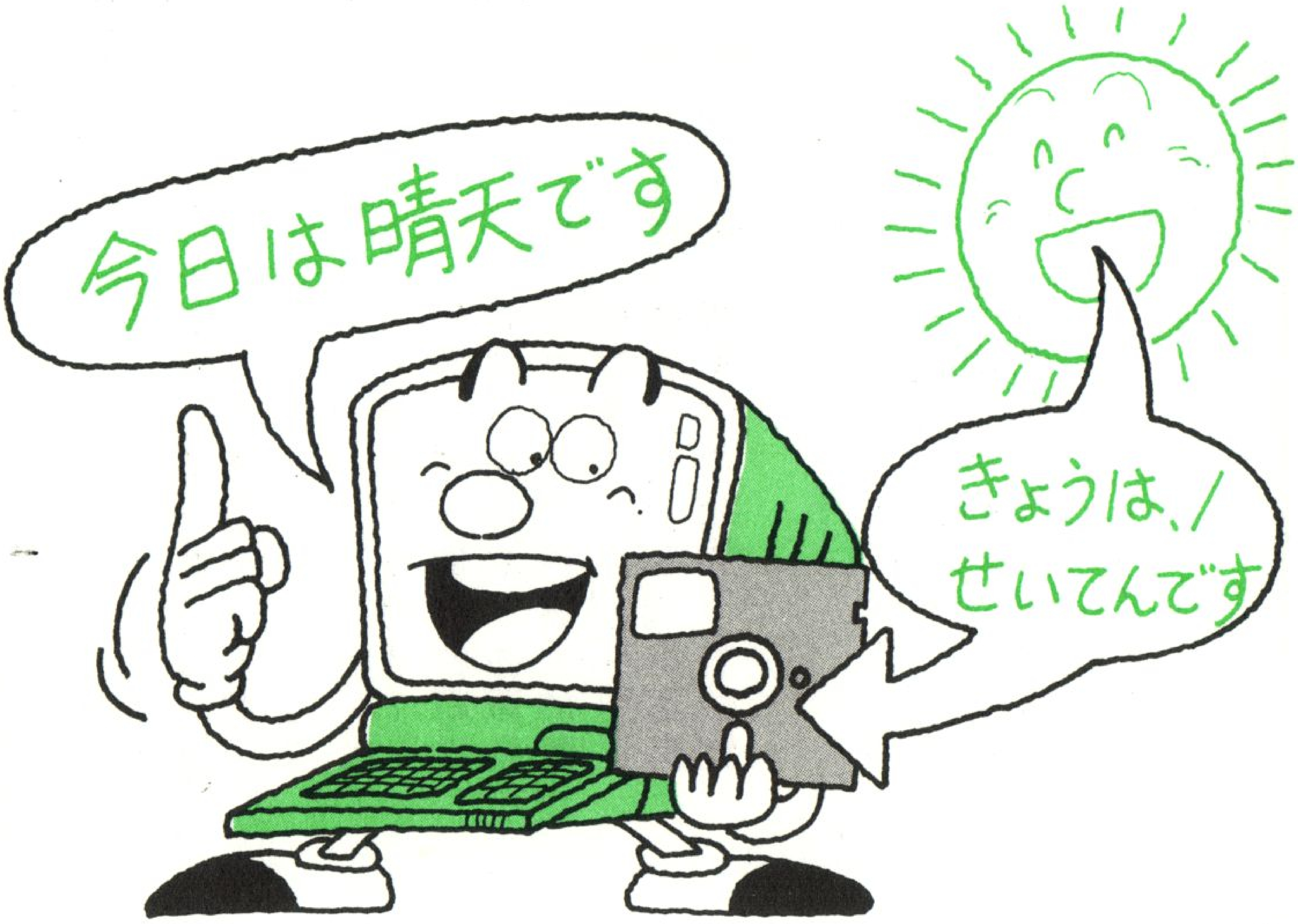
録したデータをパソコン本体内部に入力します。
データを入力する場合、データはファイルという形でディスクに記録されていますから、データを入力することを、**ファイルからの読み込み**といいます。
つまり、MS-DOSのディスク装置とディスクの操作とは、このファイルへの書き込み、あるいは、ファイルからの読み込みが完全に行えるように、書き込み、読み込みが行われるときに生じるすべてのことを管理して、それらがスムースに行えるようにするということです。
したがって、MS-DOSを使うと、MS-DOSに備わっているコマンドひとつで、ファイルへの書き込みやファイルからの読み出しを簡単に行うことができるようになります。
もちろんMS-DOSは、ディスク装置やディスクの操作ばかりでなく、パソコン全体を操作するソフトウェア（プログラム）ですから、ファイルの内容をディスプレイに表示したり、プリンタから印字して打ち出すことなどもコマンドひとつでできるようになります。

日本語をデータとして使える

MS-DOSがコマンドひとつで簡単にディスク装置やディスクの操作ができる便利なものであるとしても、入力できるデータがアルファベット、ひらがな、カタカナばかりであっては、漢字・ひらがな混りの日本語を母国語にしている私たちにとって、実際には何の役にも立ちません。
そこでMS-DOSでは、日本語を入力して漢字混りの文書で処理することができるように、漢字の辞書が備えられています。
MS-DOSに備えられている漢字辞書は、つぎの二つです。

単語／熟語辞書 文節辞書

単語／熟語辞書は、ひらがなを入力すると、入力したひらがなを、一つの漢字に変換する漢字辞



MS-DOSは日本語をデータとして使える

書です。
　たとえば、「今」と入力する場合は、「いま」と入力して漢字に変換するキーを押すと、「今」に変換して入力することができます。また「日」と入力する場合は、「にち」と入力して漢字に変換するキーを押すと、「日」に変換して入力することができます。
　このように、単語／熟語辞書は入力したひらがなを一つの漢字に変換する辞書ですから、「今日」などのように、二つ以上でできた漢字に変換することはできません。
　文節辞書は、文節でひらがなを入力して漢字に変換する辞書です。
　文節とは、つぎのように、

きょうは、／あめです。

／で区切られた「きょうは、」とか「あめです。」をいいます。
　したがって、「きょうは、」とひらがなを入力して変換キーを押すと、「今日は、」に変換して入力することができます。また「あめです。」とひらがなを

```
A>edlin 住所録
ファイルを終わりまで読込みました.
*1
     1:*竹下一男     板橋区若木町1-20-3
     2: 大山均      渋谷区渋谷5-9-4
     3: 木下太朗     港区青山2-20-6
     4: 雨宮麻衣子    世田谷区駒沢5-11-7
     5: 杉崎美代子    港区大門9-10-24
     6: 坂本龍夫     新宿区神楽坂4-8-12
     7: 中村毅      千代田区三番町5-17-14
     8: 田中一郎     大田区上池台4-7-15
     9: 浅野佳子     新宿区加賀町1-13-10
    10: 森田昌子     豊島区豊島5-17-16
*
```

MS-DOSで作成した住所録

入力して変換キーを押すと、「雨です。」に変換して入力することができます。

このように文節辞書は、文節で区切って入力したひらがなを、ひらがなと漢字混じりの文節に変換できます。もちろん、一つの漢字、二つの漢字、三つの漢字などに変換することもできます。

あに＝兄　あさ＝朝　きく＝菊
じょうげ＝上下　はいけい＝拝啓
げんき＝元気　どうそうかい＝同窓会
ゆうえんち＝遊園地

また、入力したひらがなを都道府県名や氏名などの固有名詞に変換することもできます。

とうきょうと＝東京都　しぶやく＝渋谷区
おおさかし＝大阪市　あべのく＝阿倍野区
やまだ＝山田　いちろう＝一郎

このように、MS-DOSには漢字に変換する辞書があって、日本語をデータとして使うことができますから、住所録などの名簿や文書などを作成して、ディスクに記録して保管しておき、必要に応じてデータを取り出して処理することができます。

ワープロソフト

ワープロといえば、文書とか図表などを作成するためのもので、ひらがなやローマ字を入力して漢字に変換するキーを押すと、漢字とひらがな混じりの文に変換されますから、このことを繰り返して、文書とか図表などを作成していくわけです。

MS-DOSの機能を活用した、アプリケーション・ソフトのワープロソフトもおなじで、「新・一太郎」などのようなワープロソフトのディスクを購入してきてディスク装置に挿入すると、16ビット・パソコンがワープロになります。

さきに示したように、MS-DOSでも、入力したひらがなを漢字に変換して、住所録や文書などを作成することができますが、MS-DOSの場合、入力したひらがなを漢字に変換するだけで、ワープロとして欠かせない一行の字詰めを22字詰めにしたり、32字詰めにする編集機能はありません。

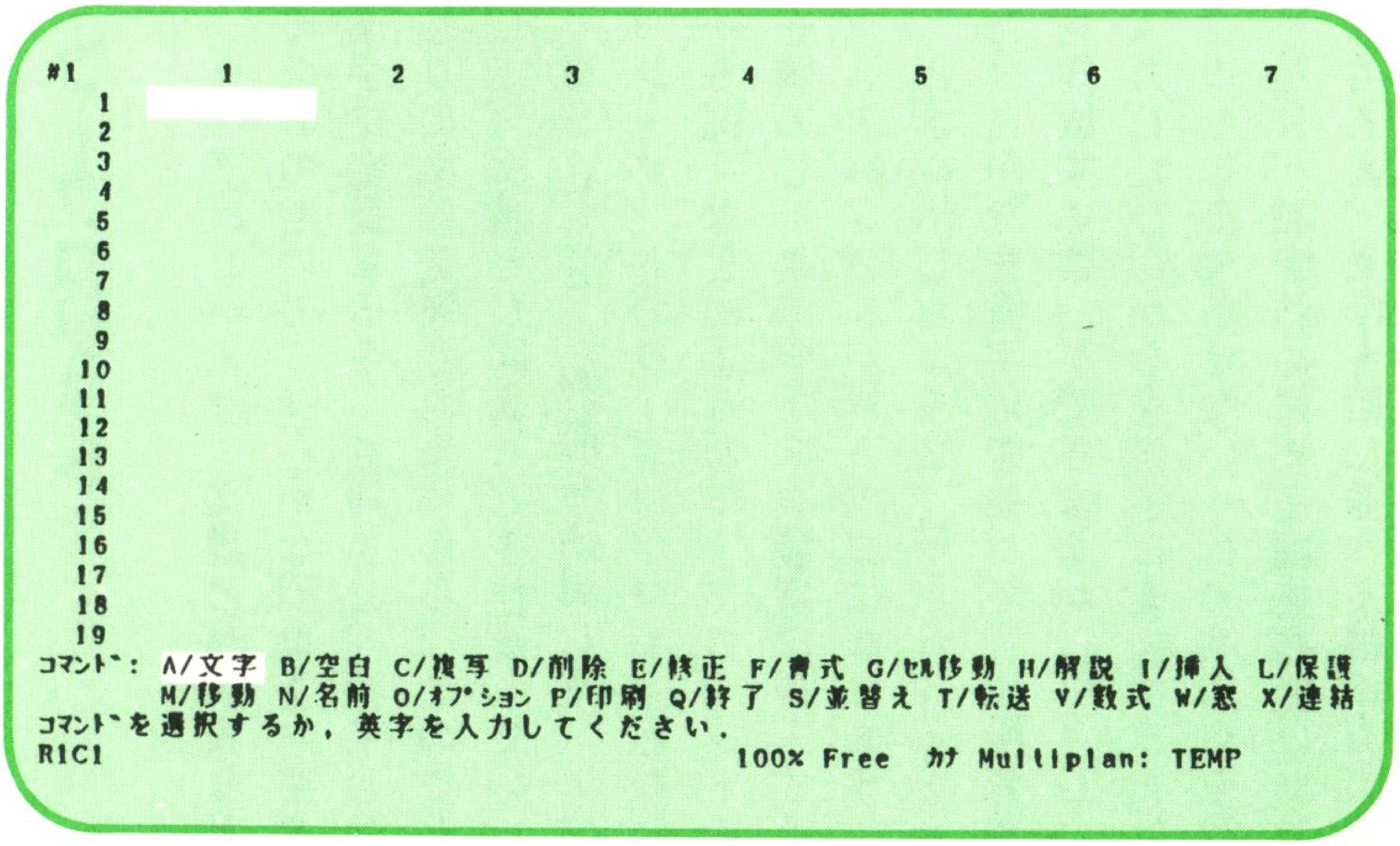
もちろん、文字を縦に並べる縦組みの機能や、文字を二段に組んだり、三段に組んだりする段組みの機能もありません。

そこで、MS-DOSが備えている機能や文節辞書をそのまま活用し、それにワープロとして必要な機能を追加したのが、アプリケーション・ソフトのワープロソフトです。

漢字を登録することができるのが特徴

ワープロソフトの優れた特徴は、必要に応じて文節辞書に漢字を登録することができるということです。

これは、MS-DOSが文節辞書に漢字を登録する機能を持っているからできるのですが、JIS第一水準や第二水準以外の漢字、文節辞書に登録されていない漢字で日頃よく使うものを文節辞書に登録できるワープロと、できないワープロとでは、どちらがワープロとして使いよいかというと、必要に応じて文節辞書に漢字を登録することができるワープロのほうが、比較にならないほど使いよいといえます。自分のよく使用する漢字を登録しておけば、効率のよい文書作成ができます。



マルチプランの画面

表計算ソフト

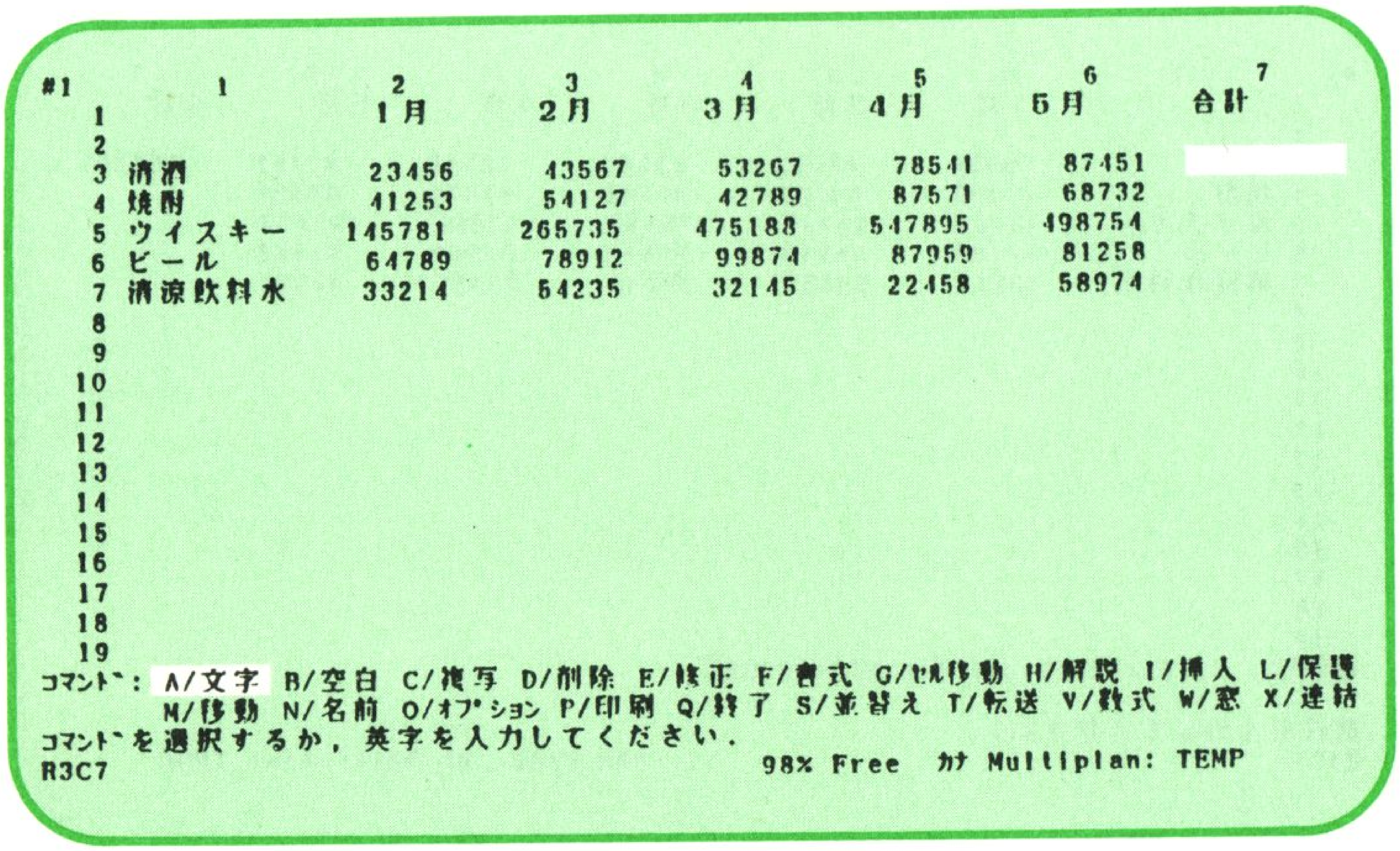
アプリケーション・ソフトの表計算ソフトには、さきに示したマルチプランのほかに、マイカルク、スーパーカルク、ロータス1-2-3などがありますが、いまでは表計算ソフトといえば、マルチプランというほど有名になっています。

マルチプランもワープロソフトとおなじように、MS-DOSの機能や単語／熟語辞書をそのまま活用して、それに表計算を行うのに必要な機能、つまり表を作成したり、計算をしたりする機能を追加したものです。

ところで、マルチプランは、上の図に示すように、画面を一枚の集計用紙とみなします。

画面の上、横方向につけられている1から7までの数字は、列を示す番号です。画面の左、縦方向につけられている1から19までの数字は、行を示す番号です。

マルチプランを実行させたとき、画面には列の



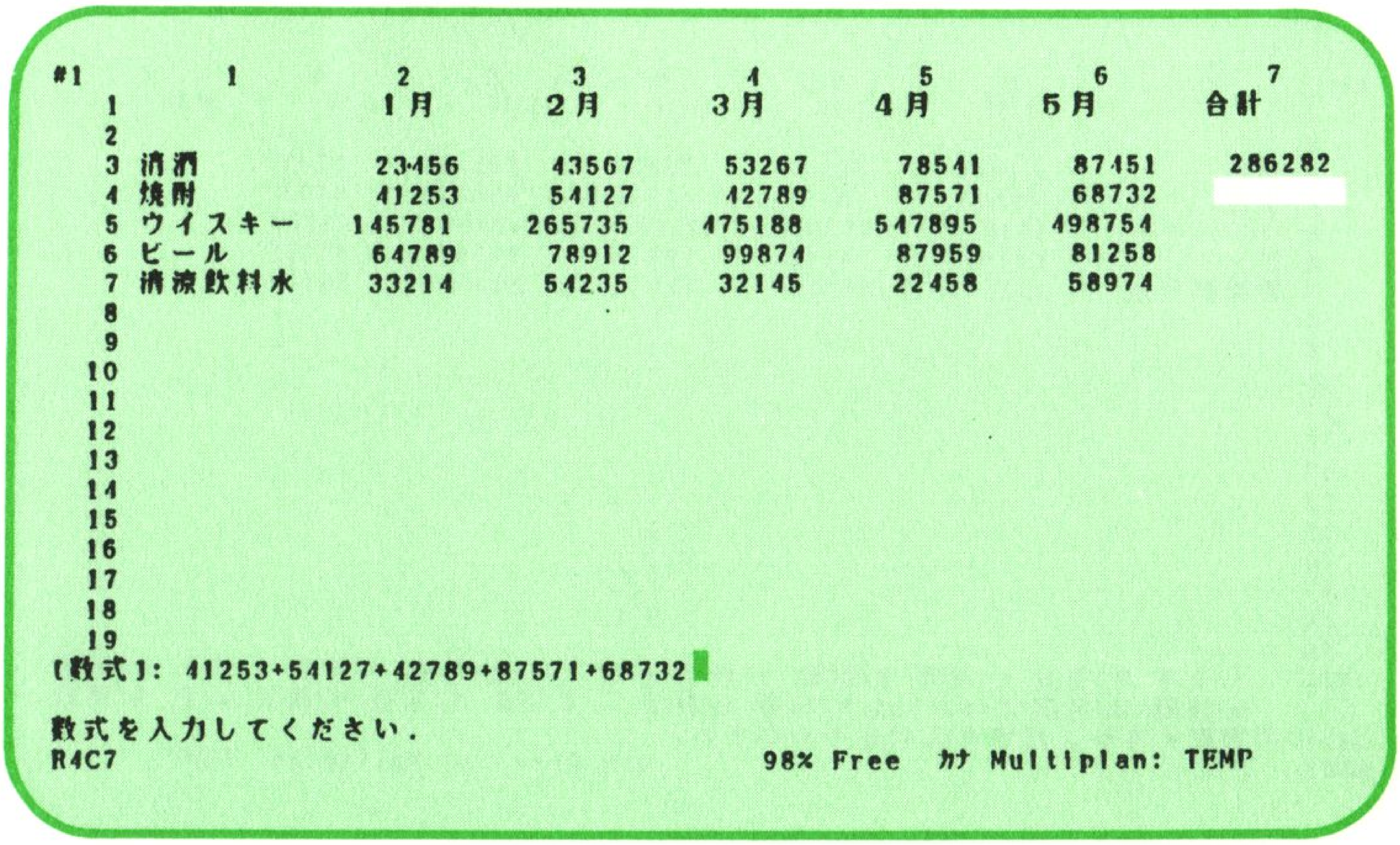
表を作成する

番号7まで、行の番号19までしか表示されませんが、画面に表示されているカーソルを右に、あるいは下に移動させると、つぎつぎに列の番号、あるいは行の番号が表示されていきます。

　このように、一枚の集計用紙（画面）は、横255列×縦4095行という非常に大きなものになっていて、この集計用紙に表を作成して集計するわけです。もちろん、この大きな集計用紙を全部使う必要はなく、必要なだけ使えばよいのです。

　たとえば、酒店の1月分から5月分までの酒類の商品別売り上げを集計するとすれば、上の図に示すように、1月から5月までの月と合計、またビール、ウイスキー、清酒、焼酎、ワインなどの商品名を入力します。罫線は引いても引かなくてもかまいません。そして、それぞれの商品の月別の売り上げを集計します。

　上の図のように、一画面に作成した表の場合は、集計するデータが一目でわかりますから、つぎの頁の図に示すように、データを入力して足していけばよいのですが、作成した表が大きい場合は、



表計算の例

表の全体が画面に表示されないので、いちいちデータを入力していったのでは、時間がかかるばかりではなく、データの入力ミスが生じて、正確な合計を求めることができません。

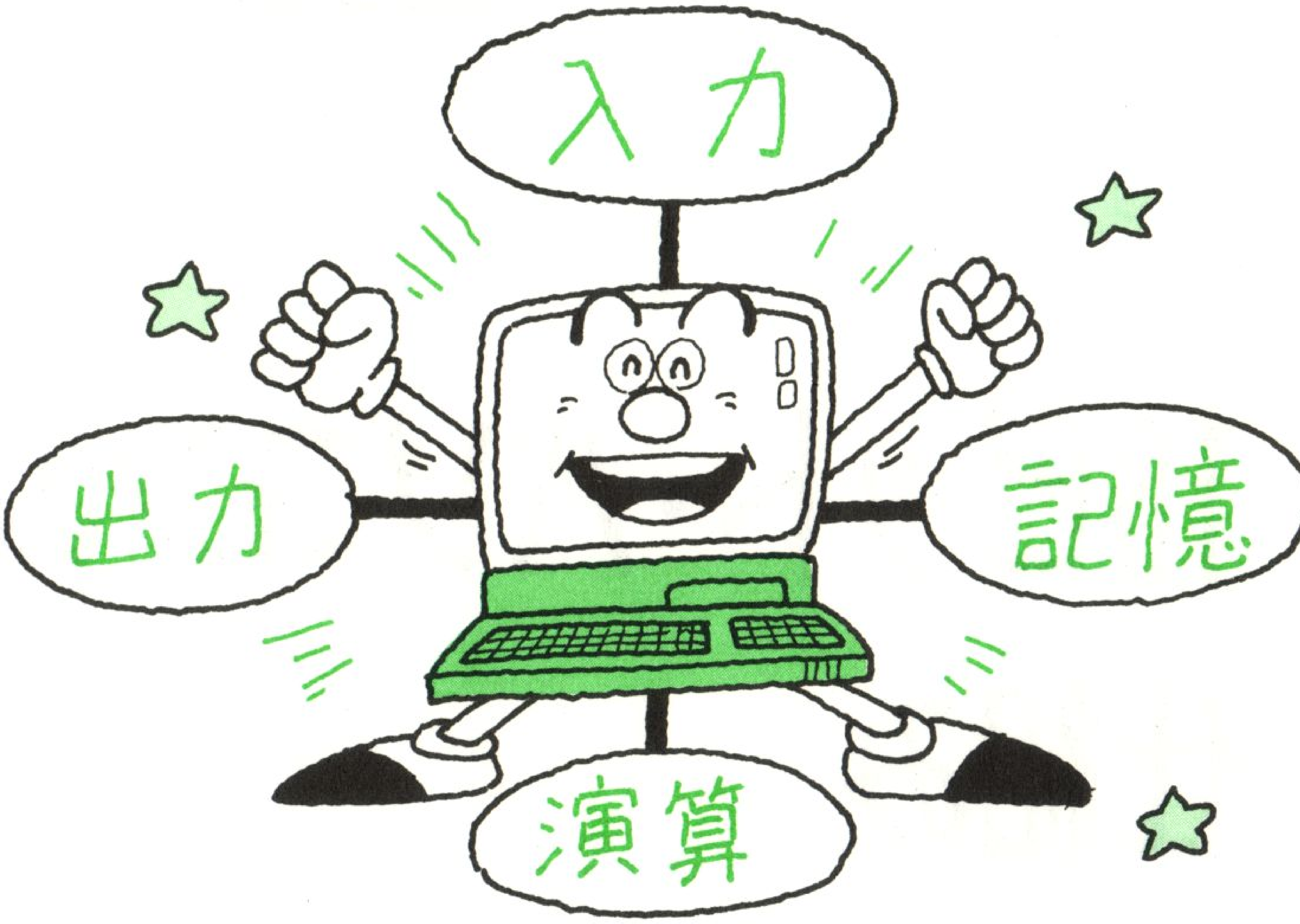
したがって、そのような場合は、「何列の何列目から何列の何列目までを足せ」と指定すると、指定された位置にあるデータのすべてが足されて、合計が求められるようになっています。

また、いったんデータの合計を求めて表を完成したあとで、もう一度データを訂正した場合は、新たにデータの合計を求めなくても、自動的に訂正したデータで合計が求められて、先に求めた合計が訂正される自動再計算という機能も備わっています。そのほか、集計するのに必要ないろいろな機能が数多く備わっていて、集計を非常に簡単なものにしています。

これらMS-DOS、また、ワープロソフトや表計算ソフトなどが使えることによって、16ビット・パソコンは、ビジネスなどに広く利用されるようになったのです。

第2章

16ビット・パソコンの仕組み



パソコンの基本的機能

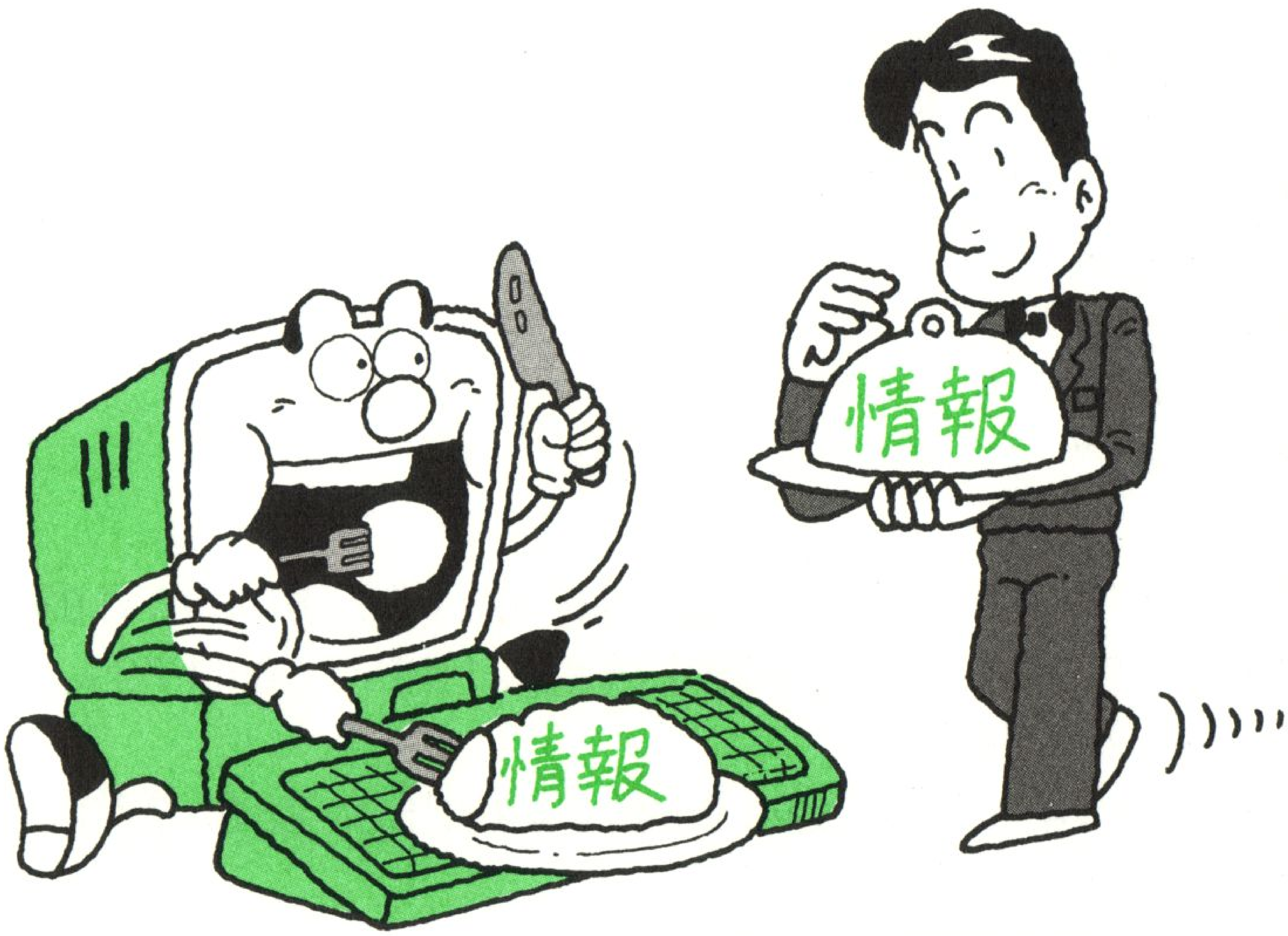
16ビット・パソコンの基本的な機能と構造

パソコンを扱ううえで知っておくべきことに、プログラミング言語であるベーシック、またベーシックによるプログラムの作り方がありますが、それらとともにパソコンの基本的な構造も知っておく必要があります。

なぜなら、単にベーシックやベーシックによるプログラムの作り方を知っているだけよりも、パソコンの基本的な構造を知っておいたほうが、より16ビット・パソコンの扱いが容易になると考えられるからです。

四つの基本的な機能

さて、16ビット・パソコンの基本的な構造を理



入力＝パソコン本体内部に情報を入れること

解するために、まず、パソコンの基本的な機能を取りあげることにします。

パソコンの基本的な機能としては、

入力（インプット）
記憶
演算
出力（アウトプット）

の四つを取りあげることができます。

もちろん、この四つの機能は、16ビット・パソコンだけにかぎったことではなく、コンピュータと名のつくものであれば8ビット機、32ビット機でも備えている機能ですが、この四つの機能が密接に組み合わさって働くことにより、パソコンはコンピュータとしての役割を果たしています。

では、これらの機能がどんな働きをするのか、入力から順に示します。

■入　力（インプット）

入力（インプット）とは、パソコンが仕事を行うために必要な情報を、キーボードなどからパソ

```
10 CLS
20 FOR I=1 TO 30
30 READ A
40 B=B+A
50 PRINT B
60 NEXT I
70 END
80 DATA 10,20,30,40,50,60,70,80,90,100
90 DATA 110,120,130,140,150,160,170,
180,190,200
100 DATA 210,220,230,240,250,260,270,
280,290,300
```

- 10 CLS ← 画面を消去せよ
- 20 FOR I=1 TO 30 ← 行番号30から50までを30回繰り返せ
- 30 READ A ← データを読み取れ
- 40 B=B+A ← 読み取ったデータを加えよ
- 50 PRINT B ← 加えた結果を表示せよ
- 60 NEXT I ← 行番号20に戻れ

（10〜60）プログラムは命令の集り

（80〜100）プログラムで処理するデータ

プログラムとデータ

コンの本体内部に入れてやることです。

この場合の情報とは、プログラムとデータのふたつです。

プログラムとは、上に示すように、パソコンが仕事をしていくための手順を示したもので、「―せよ」という命令の集りです。

データとは、プログラムで処理するもののことです。

ふつう、パソコンの本体内部に情報を入力するには、キーボードのキーを叩いて行いますが、最近ではほとんどのパソコンは、一度入力した情報を、ディスク装置に挿入したフロッピーディスクに記録して保管することができます。そして、必要に応じてフロッピーディスクに記録した情報をパソコン本体に入力するという方法も取ります。

キーボードのキーを叩いて情報をパソコン本体の内部に入力したときはキーボードが、また、フロッピーディスクに記録した情報を入力するときは、ディスク装置が入力装置となります。



記憶＝パソコン本体内部の記憶場所に情報をしまうこと

記憶

記憶とは、パソコンの本体内部にある記憶場所に、入力された情報をしまっておくことです。

この場合の情報には、二種類あります。

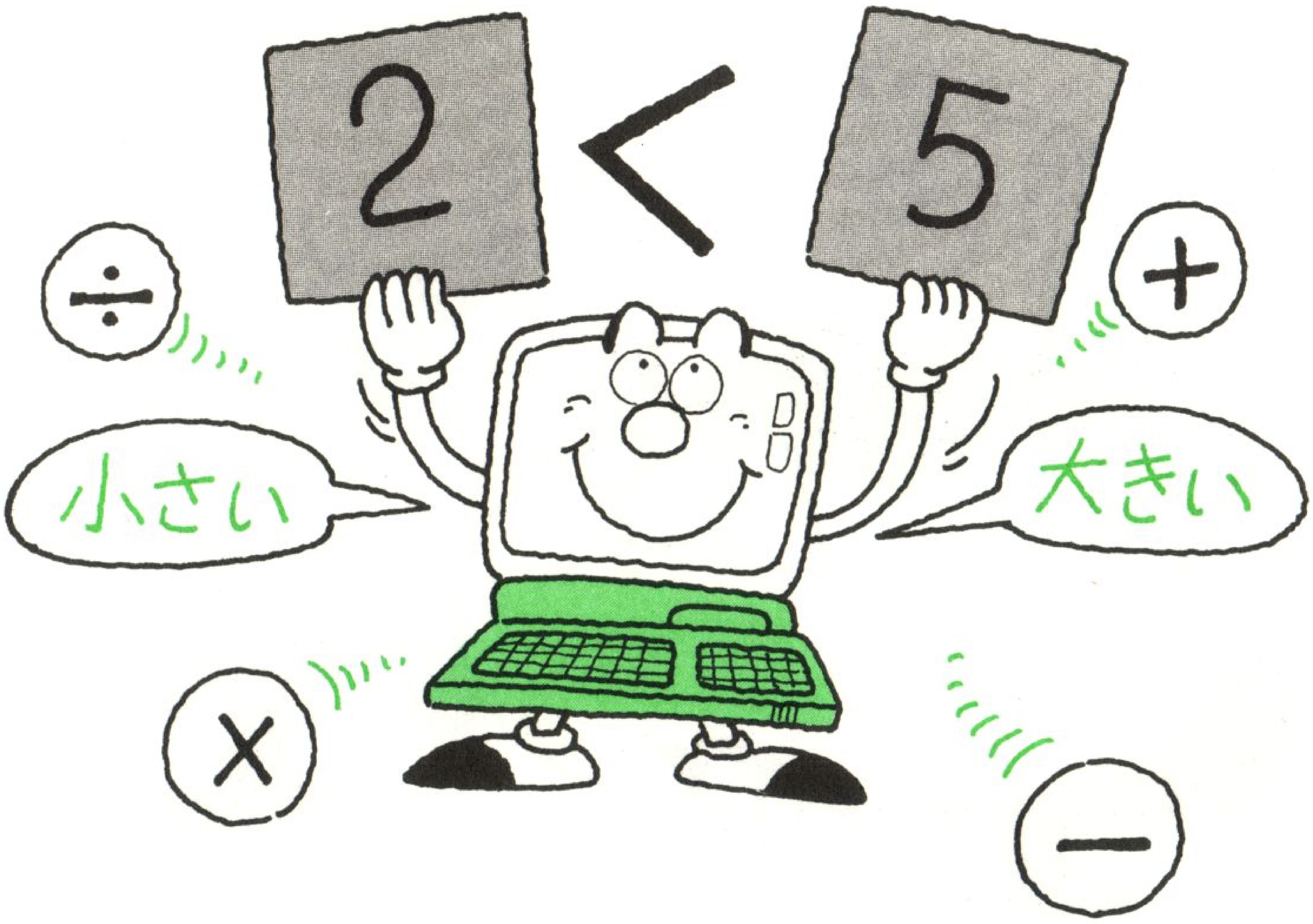
一つはパソコンの内部に入力されたプログラムやデータです。

パソコンは、入力されたプログラムやデータを、いったん本体内部にある記憶場所に記憶したあと、それをひとつひとつ取り出して処理していくようになっています。

このように、プログラムやデータを本体内部にある記憶場所に記憶したうえで処理していく方法を、プログラム内蔵方式、あるいはプログラム記憶方式といいます。

ところで、パソコンの本体内部にある記憶場所に記憶される情報には、もうひとつ、データを処理した結果があります。

ただし、データを処理した結果の場合は、必ず記憶場所に記憶されるわけではありません。プログラムのなかに記憶場所が指定されている場合は、



演算＝計算と判断を行うこと

指定された記憶場所に記憶されますが（118頁参照）、プログラムのなかに記憶場所が指定されていない場合は、データを処理した結果は、記憶場所とは別のレジスタ部というところに記憶されたままになります（37頁参照）。

演　算

演算とは、足し算、引き算、掛け算、割り算を行うことですが、この四つを総称して「四則演算」といいます。

また、「四則演算」を行うだけではなく、数を比較して、どちらの数が大きいか、小さいかなどを判断する「判断」も、演算に含まれます。

このように質の異なる「四則演算」と「判断」を一緒にして「演算」という理由は、この二つが異なった場所で行われるのではなくて、演算回路（34頁参照）というおなじ場所で行われるからです。

このようなことから、「四則演算」と「判断」の二つをひっくるめて、「演算」といっているのです。



出力＝パソコンの本体内部の記憶場所から情報を送り出すこと

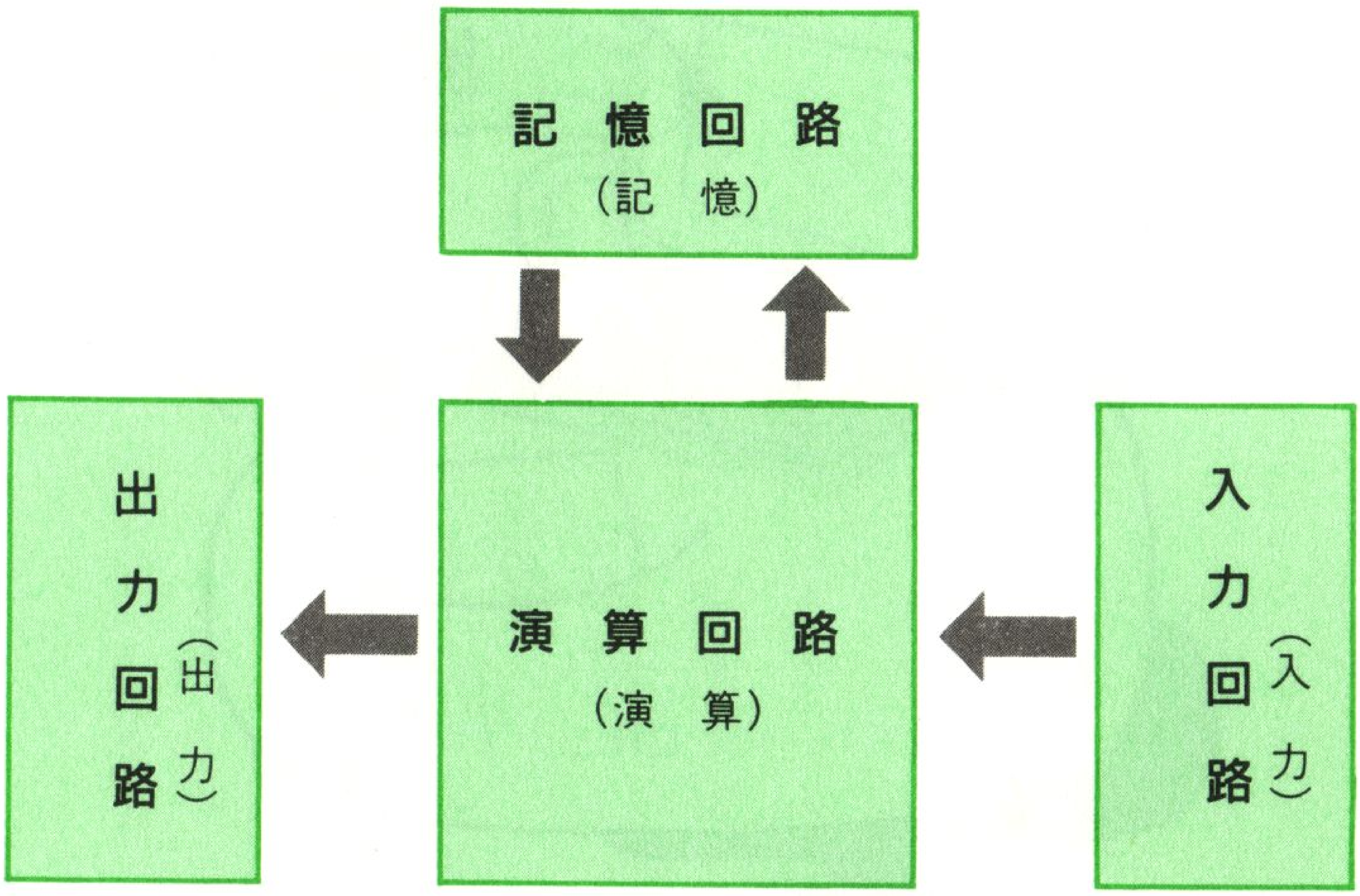
出　力（アウトプット）

出力（アウトプット）とは、パソコンの本体内部にある記憶場所、あるいはレジスタ部に記憶されている情報を、ディスプレイなどに向けて送り出すことをいいます。

たとえば、データを処理した結果がパソコンの記憶場所などに記憶されたままでは、私たちは処理した結果がどのようになったかを知ることができません。

そこで、プログラムのなかにディスプレイに向けて出力させる命令を書いておいて、ディスプレイに向けて出力させるようにすると、ディスプレイに処理した結果が表示されるわけです。

このように、情報をディスプレイに向けて出力させるなら、ディスプレイが出力装置、プリンタに出力させるなら、プリンタが出力装置となります。また、情報をフロッピーディスクに記録させるために出力させるのなら、フロッピーディスクを挿入しているディスク装置が出力装置となります。



パソコンの基本的な構造

基本的な構造

パソコンのもつ入力・記憶・演算・出力という基本的な機能がわかったところで、つぎはパソコンの基本的な構造です。

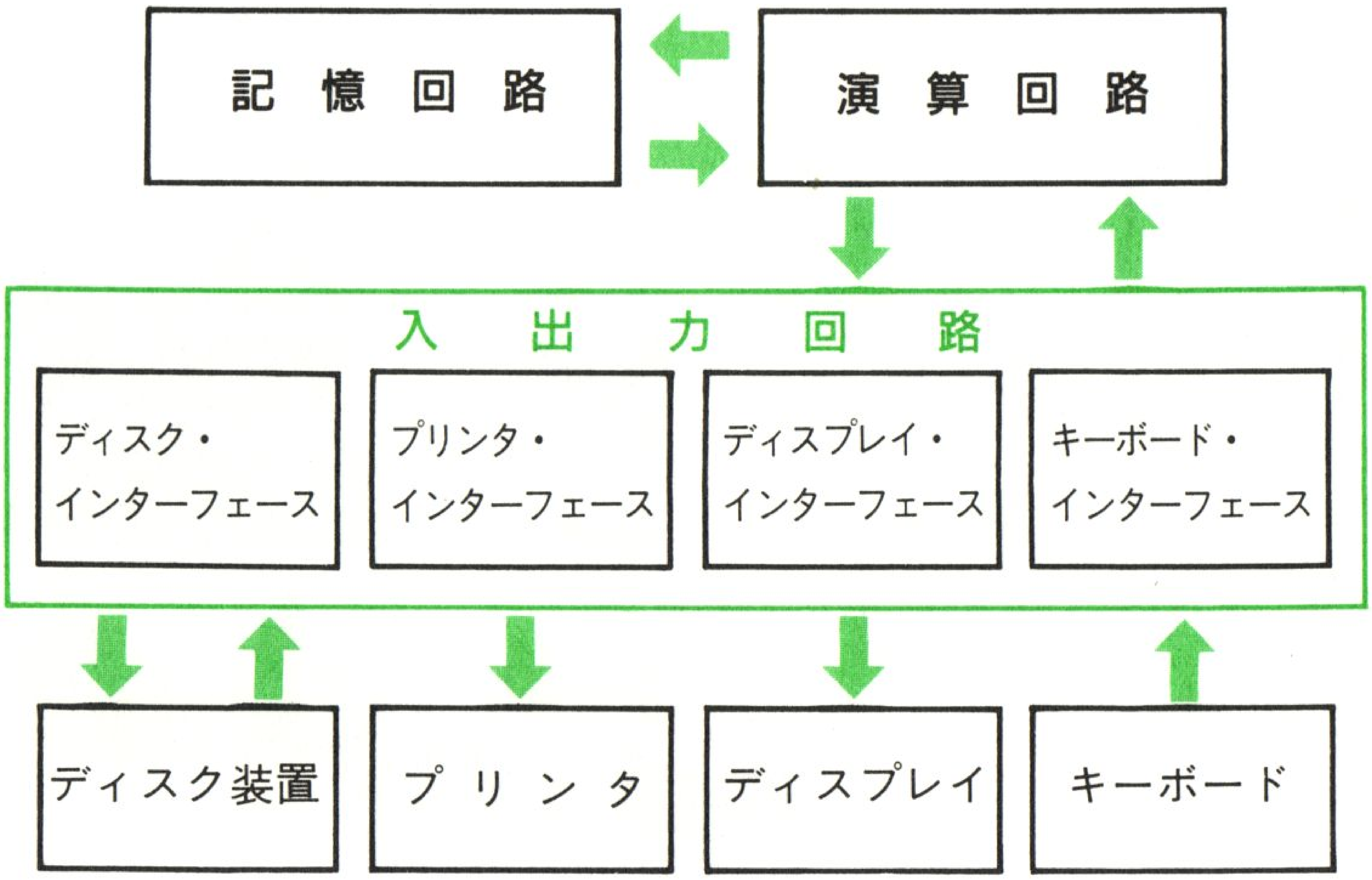
パソコンのカバーをあけて本体をみてみると、基盤のうえに黒くて細長い形をしたＩＣやＬＳＩがぎっしりと並んでいます。

これらのＩＣとＬＳＩを役割り別に大きく分けてみると、四つのグループに分けることができます。

四つのグループとは、入力を受け持つグループ、記憶を受け持つグループ、演算を受け持つグループ、出力を受け持つグループです。

入力を受け持つグループを**入力回路**、記憶を受け持つグループを**記憶回路**、演算を受け持つグループを**演算回路**、出力を受け持つグループを**出力回路**といいます。

このうちの入力と出力の回路を、**インターフェ**



■外部装置と演算回路は入出力回路で結ばれている■

ースと呼んでいます。また、入力と出力の回路を一緒にして、**入出力回路**といいます。

パソコンの基本的な構造は、この四つの回路から成り立っていて、まえの項で示した入力、記憶、演算、出力の四つの基本的な機能は、つぎに示すように、

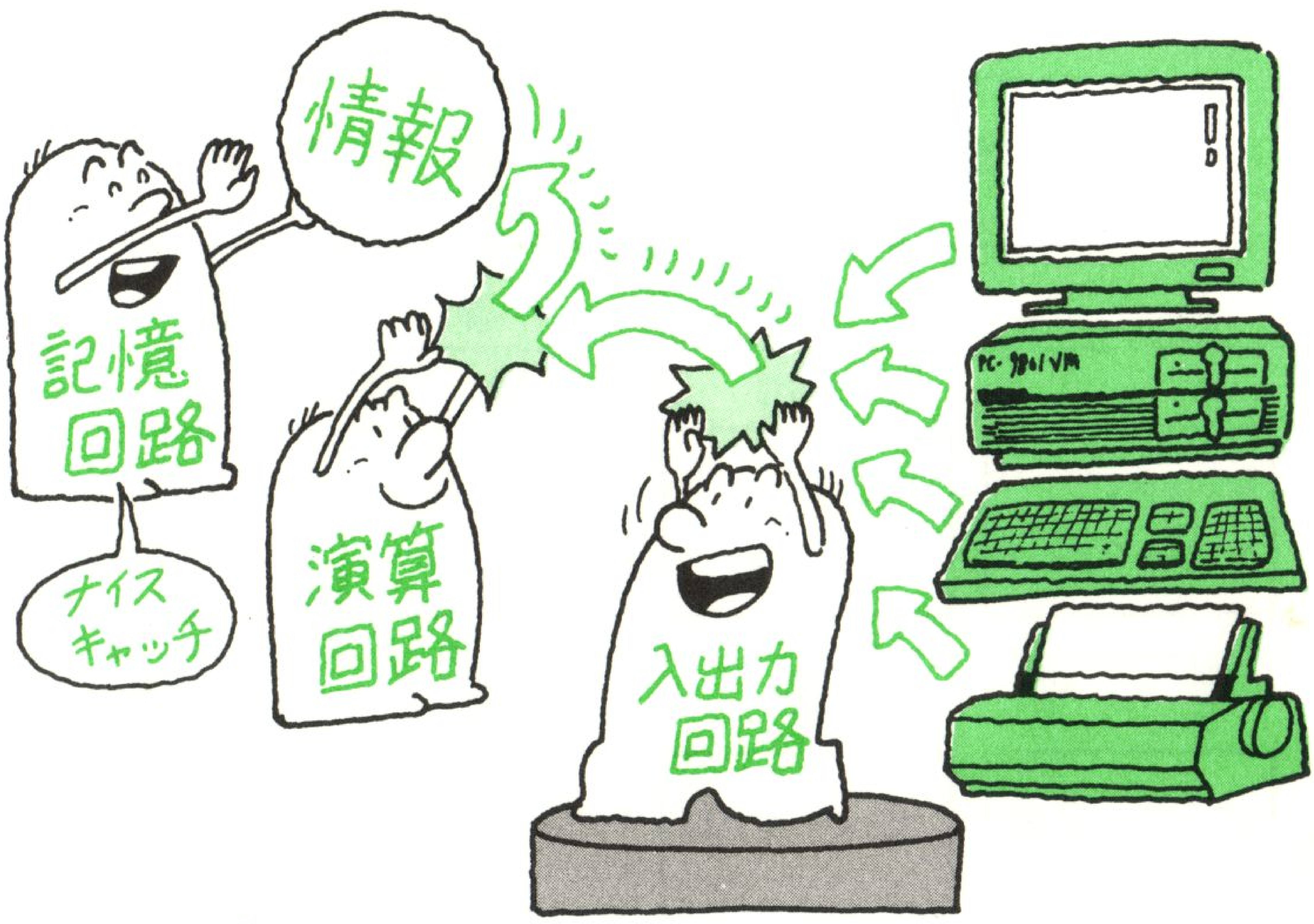
入力＝入力回路
記憶＝記憶回路
演算＝演算回路
出力＝出力回路

四つの回路がそれぞれ受け持っています。

では、入力回路と出力回路を一緒にした、入出力回路の働きから取りあげます。

■入出力回路（インターフェース）

パソコンの本体内部の演算回路と外部装置、つまり情報を入力するキーボード、情報を出力するディスプレイやプリンタ、フロッピーディスクに記録されている情報を入力したり、あるいは情報をフロッピーディスクに記録したりするディスク



インターフェースは情報の受け渡しがうまくできるように働く

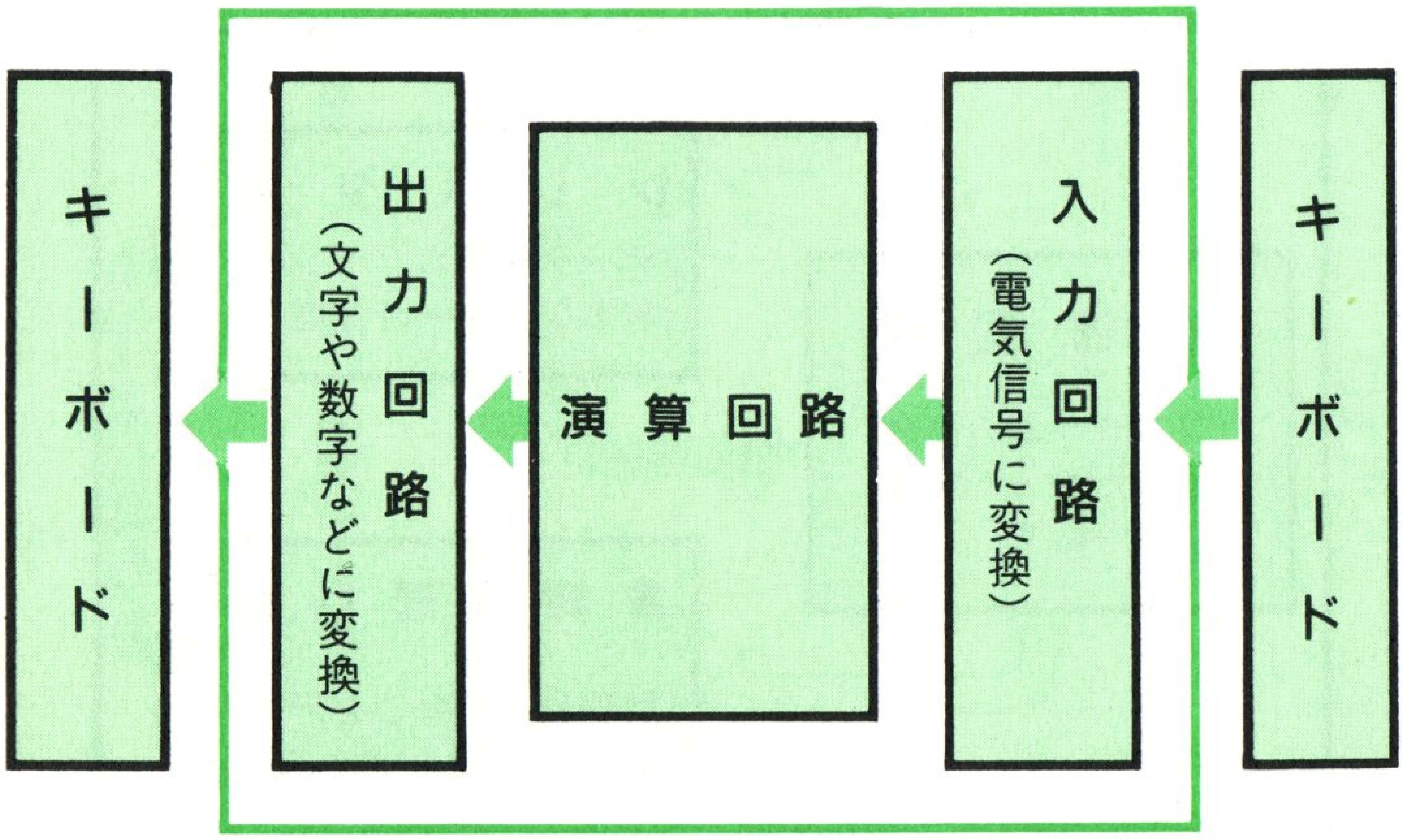
装置とは、前頁の図に示すように、入出力回路、つまりインターフェースを間にはさんで結ばれています。

したがって、たとえばキーボードのキーを叩いて情報を入力したときは、入力した情報はキーボード・インターフェースを通って演算回路に受け取られ、それから記憶回路に記憶されます。

また、記憶回路に記憶されている情報をディスプレイに出力するときは、記憶回路に記憶されている情報は演算回路に受け取られ、それからディスプレイ・インターフェースを通って、ディスプレイに出力されます。

「インターフェース」のもとの意味は、ふたつの噛み合わさった歯車が接する部分のことです。

パソコンのインターフェースもおなじように、本体内部の演算回路とキーボードなどの外部装置との間にあって、両者がうまく噛み合うことができるような働きをしているのです。つまり、本体内部の演算回路とキーボードなどの外部装置との間を、うまく情報の受け渡しができるように仲介



入力回路で電気信号に、出力回路で文字などに変換

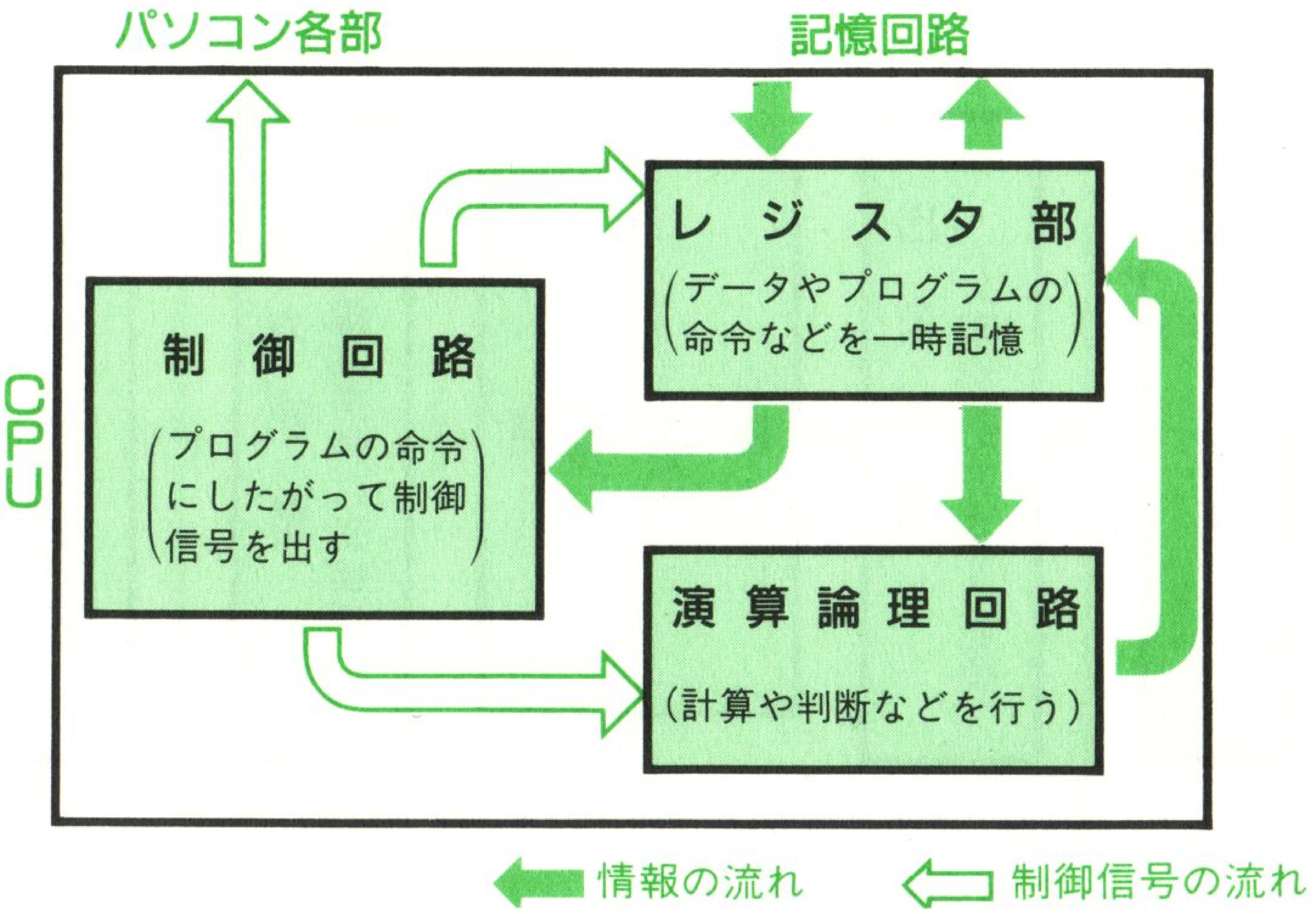
しているのです。

このようにインターフェースが必要となる理由は、演算回路の処理スピードがものすごく速くて、キーボードなどの外部装置の処理スピードでは追いつくことができないからです。

たとえば、「A」という文字を入力するためにキーボードのAのキーを叩いたとすると、Aのキーに指がふれて離れる間に、演算回路は「A」の文字を千個も二千個も取り込んでしまうほど、その処理スピードは速いのです。そこで、演算回路とキーボードの間にインターフェースを置いて、キーボードのAのキーを一回叩いたら、演算回路が「A」の文字を一個取り込むように働いているわけです。

さて、キーボードのキーを叩いて入力する文字や数字、あるいは記号などは、キーボード・インターフェースを通るときには、電気信号に変換されてパソコン本体内部に入っていきます。

たとえば、Aのキーを叩いたとすると、「A」という文字は、「A」を意味する電気信号に変換され、



CPUの構造

また①のキーを叩くと、「1」という数字は「1」を意味する電気信号に変換されて、パソコン本体内部に入っていくわけです。

　また、ディスプレイに出力される場合は、ディスプレイ・インターフェースを通るときに電気信号が意味する文字や数字に変換されて、ディスプレイに出力されるようになっています。

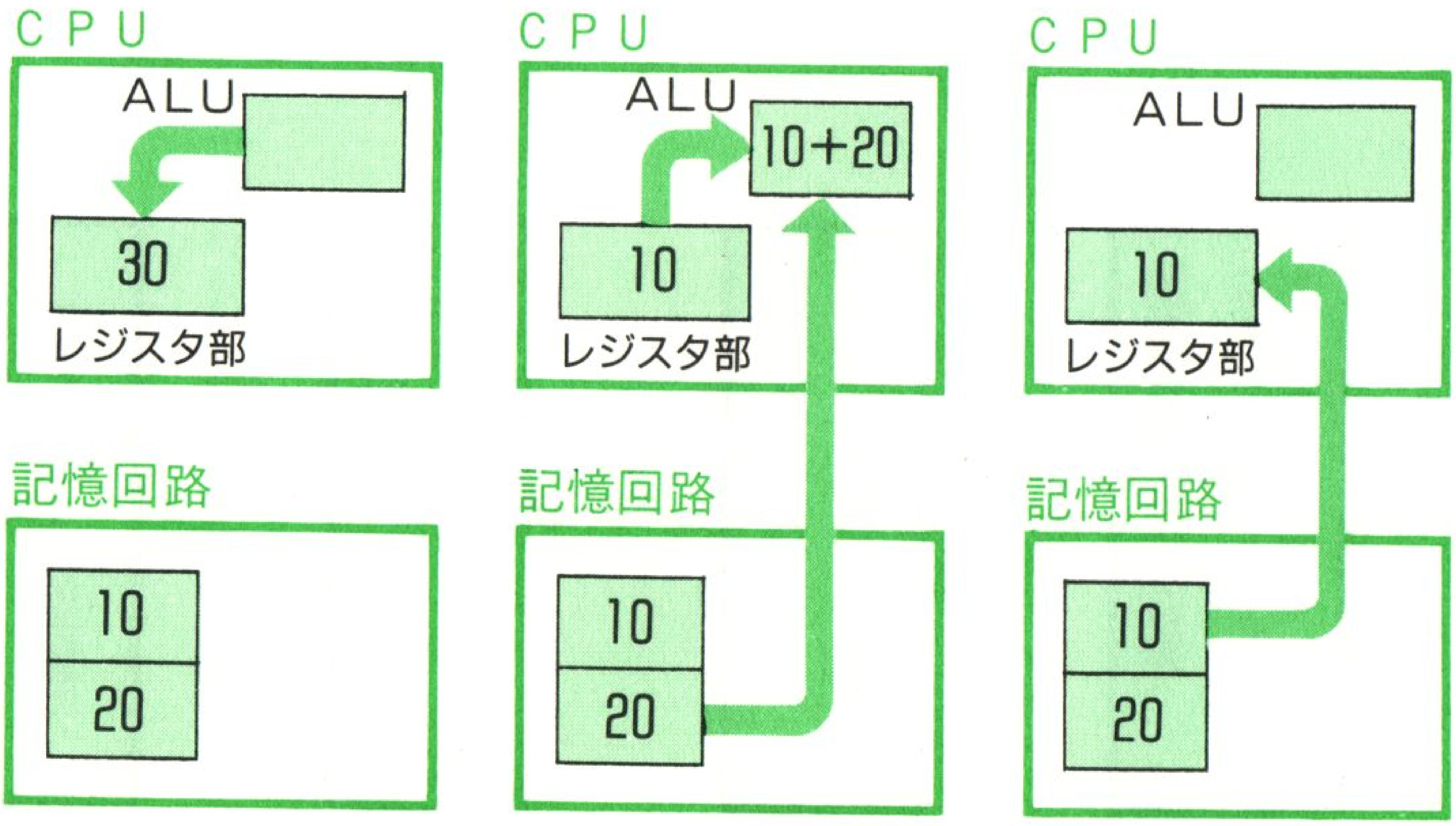
■演算回路

　演算回路は、パソコンの心臓部といわれるところで、ふつうCPU（セントラル・プロセシング・ユニット）といっています。

　この演算回路で、四則演算や数の大小の判断を行ったり、またパソコン全体が、ひとつの目的に向かって働くようにコントロール（制御）しています。

　さて、CPUは大きく分けて、上の図に示すように演算論理回路（ALU）、制御回路、レジスタ部の三つの回路で構成されています。

　この三つの回路は複雑に噛み合わさって働いているので、別々に切り離して説明することはむず



①レジスタ部に足される数10が取り込まれる

②つぎに足す数20が取り込まれて、10と20がＡＬＵに送り込まれて足される

③足し算の結果の30はレジスタ部に記憶される

演算論理回路（ALU）の働き

かしいのですが、説明する都合上、ひとつひとつ取りあげていくことにします。

① 演算論理回路（ALU）

演算論理回路は、ふつうALU（アリスメティック・ロジック・ユニット）といっています。

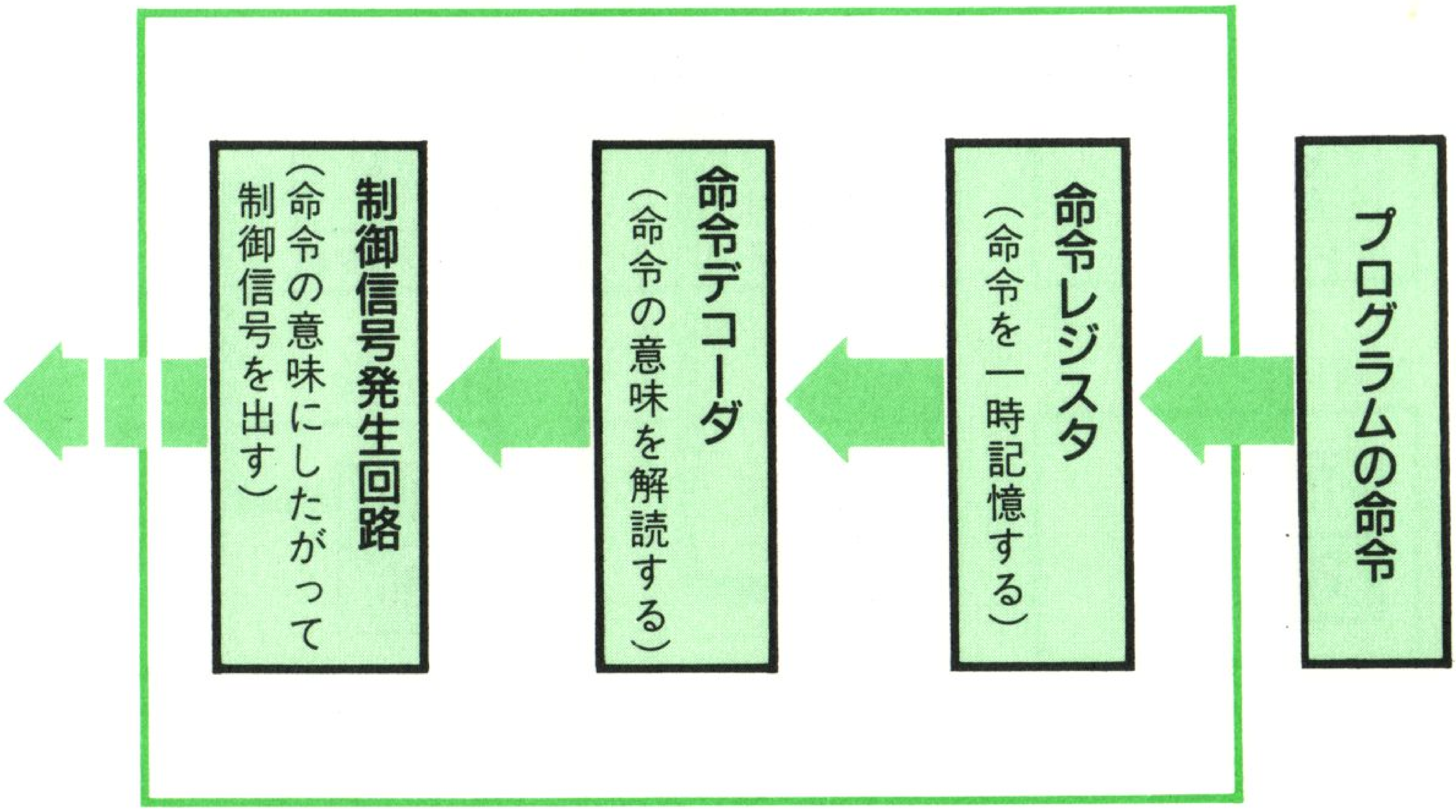
このALUで、四則演算や数の大小の判断などを行います。

たとえば、プログラムに「10＋20を行え」という命令があるとすると、上の図に示すように、CPUは命令にしたがって、まず記憶回路に記憶されている足される数「10」を、記憶回路からCPU内部にあるレジスタ部に取り込みます。

つぎに、CPUは「足せ」という命令で、記憶回路に記憶されている足す数「20」を記憶回路から取り込み、さきにレジスタ部に取り込んだ「10」と一緒にALUに送り込みます。

ALUは、「10」と「20」が送り込まれてくると「10＋20」を行います。

その結果の「30」はレジスタ部に送られて記憶されます。



制御回路の働き

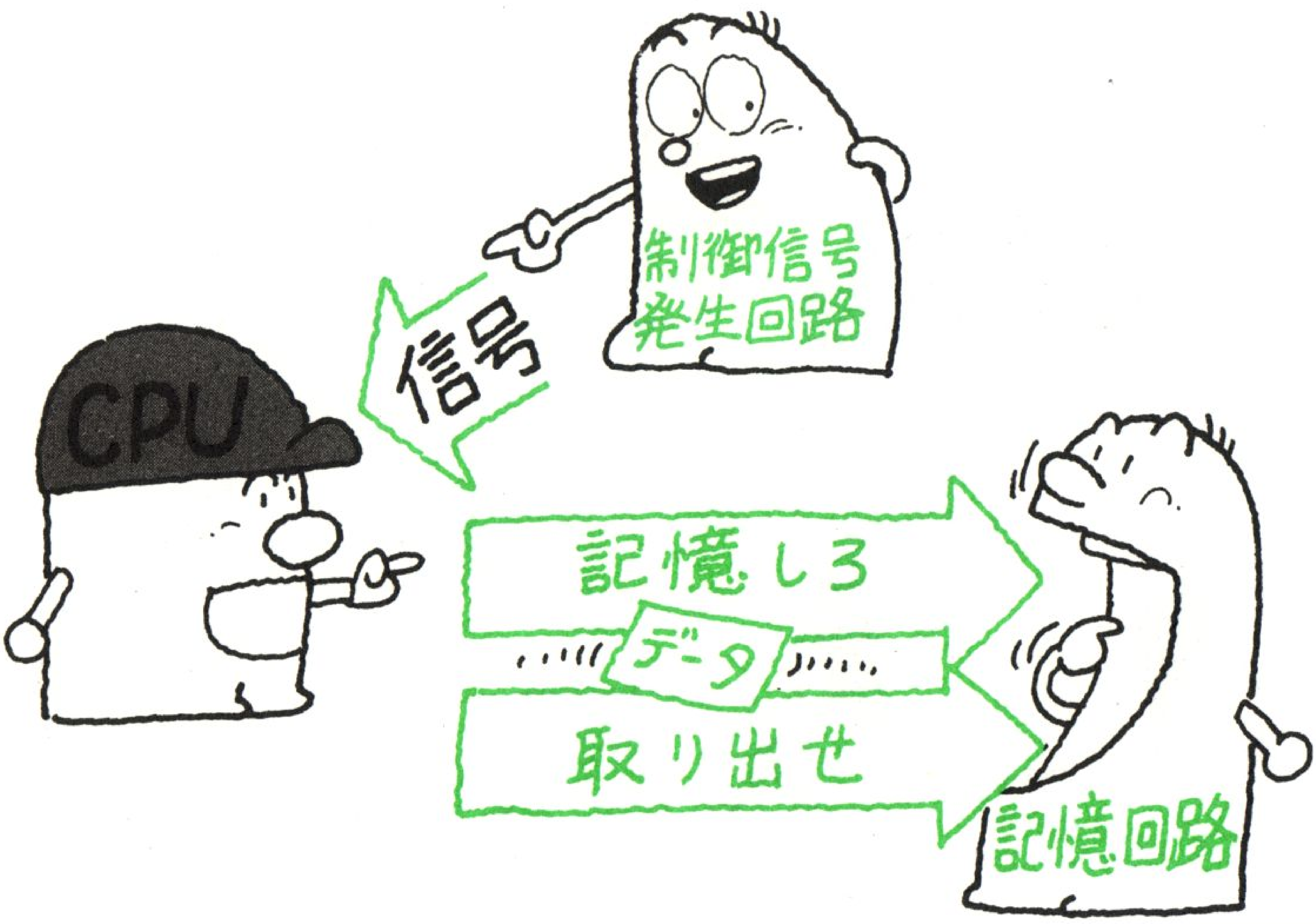
これは足し算の例ですが、足し算以外の場合もおなじようにして計算が行われます。また、数の大小の判断の場合もおなじです。

ところで、レジスタ部に記憶された「30」は、プログラムのなかに「10＋20」の結果を記憶回路に記憶させる命令があると、記憶回路に記憶されます。またプログラムのなかに、ディスプレイに出力させる命令があると、ディスプレイに向けて出力されます。

② 制御回路

制御回路は、パソコンがプログラムの命令を忠実に実行するために、プログラムの命令にしたがってパソコン本体内部の各部に指令を出すところで、上の図に示すように、命令レジスタ、命令デコーダ、制御信号発生回路の三つの部分でできています。

命令レジスタ……命令レジスタは、記憶回路から取り込んだプログラムの命令を一時記憶しておくところで、ここに一時記憶されたプログラムの命令は、つぎに命令デコーダに送られます。



制御信号にしたがってパソコン各部が働く

命令デコーダ……命令デコーダは、命令レジスタから送られてきたプログラムの命令がどんな意味の命令であるかを解読するところで、ここでプログラムの命令の意味を解読すると、その解読した意味を制御信号発生回路に伝えます。

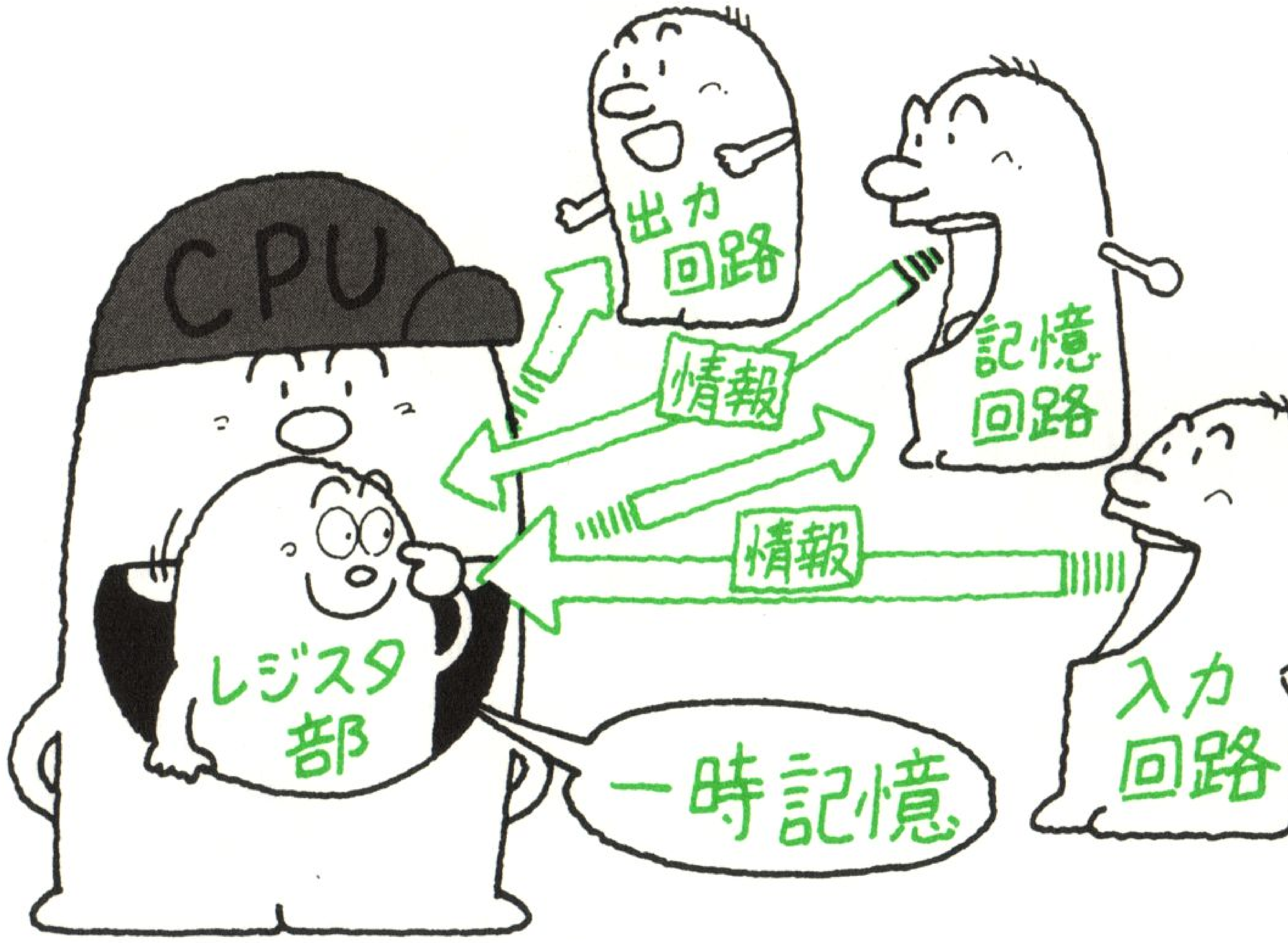
制御信号発生回路……制御信号発生回路は、CPUの内部やパソコンの各部に制御信号を出します。

制御信号発生回路から出された制御信号にしたがって、CPUの内部やパソコンの各部が秩序正しく作動して、記憶回路からデータを取り出して計算を行ったり、計算の結果を記憶回路に記憶したり、また、計算の結果をディスプレイに表示するために、出力回路に向けて出力したりします。

③　レジスタ部

レジスタ部は、記憶回路や入力回路などから、CPUの内部に取り込んだ情報を、一時記憶しておくところです。

記憶回路からCPUのなかに取り込んだ情報がプログラムの命令であれば、一時記憶したあと、



レジスタ部はCPU内部に取り込んだ情報を一時記憶する

命令デコーダで命令の意味を解読するために命令レジスタに送ります。

記憶回路からCPUのなかに取り込んだ情報が計算するためのデータであったり、数の大小を判断するためのデータであれば、一時記憶したあと、計算や数の大小の判断を行うALUに送ります。ALUでデータを処理した結果も、ここに一時記憶されます。

また、キーボードなどの外部装置から入力された情報は、入力回路を通ってCPUの内部に取り込まれますが、CPUの内部に取り込まれた情報は、このレジスタ部に一時記憶されます。

それから、記憶回路に向けて送り出されて記憶されます。

記憶回路に記憶されている情報をディスプレイなどに出力する場合も、記憶回路に記憶されている情報はCPUの内部に取り込まれますが、その場合もCPUの内部に取り込まれた情報は、このレジスタ部に一時記憶されてから、出力回路に向けて送り出されます。

```
h]d0000,010f
0000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0080 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0090 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

記憶回路のなかはいくつもの記憶場所にわかれている

■記憶回路

記憶回路は、ふつうメモリといい、入力されたプログラムやデータ、またデータを処理した結果などを記憶しておくところです。

この記憶回路のなかは、上の図に示すように、いくつもの記憶場所に細かく分けられています。この図は、PC－9801VFの記憶回路のなかの一部を、直接画面に表示させたものです。

この図を見るとわかるように、FFが数多く並んでいます。

このFFは、記憶場所になにも記憶されていない状態のときに表示されるものですが、ひとつのFFが、ひとつの記憶場所を示しています。

記憶回路にプログラムやデータを記憶するときは、この記憶場所を必要なだけ使って記憶します。

たとえば、「10＋10」を行って、その結果をディスプレイに表示させる、

10 PRINT 10+10

というプログラムの場合であると、13個の記憶場所を使って記憶します（59頁参照）。

0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D	002E	002F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

メモリ番地は、ひとつひとつの記憶場所につけられている

① メモリ番地

ところで、この細かく分けられた記憶場所には番号がつけられています。この番号のことをメモリ番地といいます。

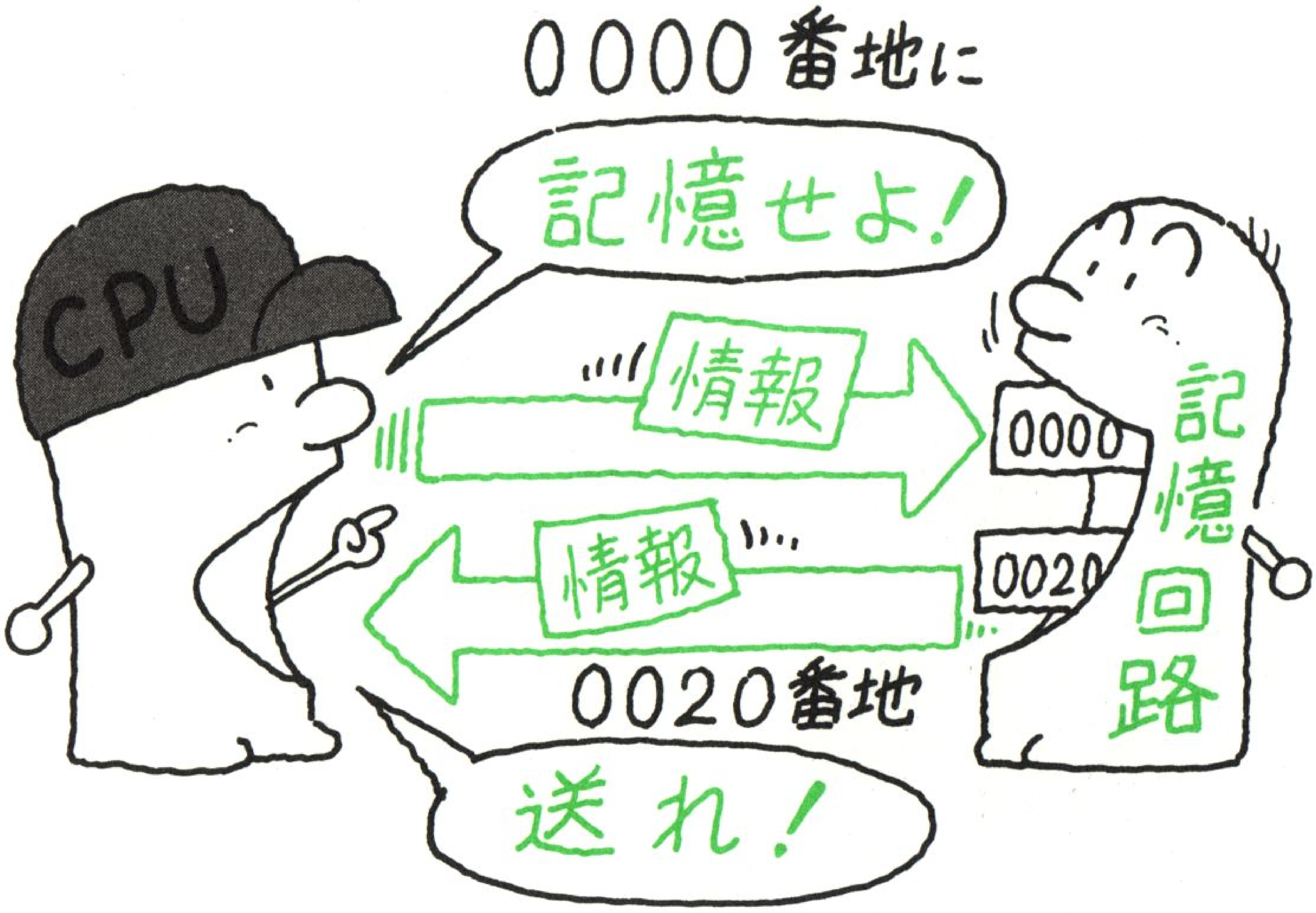
まえの頁の図の左側につけられている0000とか0010とかがメモリ番地です。

0000や0010などは16進数という数で(48頁参照)、これを私たちがふだん使っている10進数になおすと、0000は0、0010は16、0030は48ということになります。

このようにみると、メモリ番地は16おきにつけられているようにみえますが、画面に記憶回路を直接表示させたとき、16おきに表示されるのであって、実際には上の図に示すように、メモリ番地は、ひとつひとつの記憶場所ごとにつけられています。

② メモリ番地の役割

ひとつひとつの記憶場所にそれぞれメモリ番地がつけられているのは、CPUが入力されたプログラムやデータを指定の記憶場所に記憶させたり、



CPUはメモリ番地を指定して情報を出し入れする

また、記憶させたプログラムやデータを取り出したりするためです。

つまりＣＰＵは、入力されたプログラムやデータを記憶回路に記憶していく場合、ＣＰＵ内部の制御信号発生回路から「記憶せよ」という制御信号を出すと同時に、記憶場所のメモリ番地を指定するアドレス信号を出して、アドレス信号で指定したメモリ番地の記憶場所にプログラムやデータを記憶していくわけです。

また、ＣＰＵが記憶回路に記憶したプログラムやデータを取り出す場合も、制御信号発生回路から「送れ」という制御信号を出すと同時に、記憶場所のメモリ番地を指定するアドレス信号を出して、アドレス信号で指定した記憶場所からプログラムやデータを取り出していきます。

といっても、ＣＰＵが自由にメモリ番地を指定してプログラムやデータを記憶させたり、どの記憶場所にどんなプログラムやデータが記憶されているかをおぼえていて、その記憶場所からプログラムやデータを取り出すわけではありません。



記憶容量とは、記憶回路に記憶できる情報量

入力されるベーシックのプログラムやデータは記憶回路のなかのどのメモリ番地から記憶していくか、あらかじめ決められていて、自動的にそのメモリ番地から順に記憶されていき、また、取り出すときも、そのメモリ番地から順に取り出す仕組みになっているのです。

ただ、CPUが入力されるプログラムやデータを記憶場所に記憶していったり、記憶場所からプログラムやデータを取り出す場合、ひとつひとつメモリ番地を指定して行うようになっているということです。

③ 記憶容量

パソコンは、プログラムやデータが入力されたからといって、記憶回路に無限に記憶できるわけではありません。記憶回路にいくつの記憶場所があって、どれだけ情報を記憶することができるのか、パソコンの機種ごとによって決められています。

このように、パソコンが記憶することができる情報量のことを記憶容量といいます。

2進数	10進数
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

●＝電気信号が流れている
●＝電気信号が流れていない

2進数

④ バイトとビット

この記憶容量を表わすには、バイト(B)、あるいはキロバイト(KB)という単位を使います。

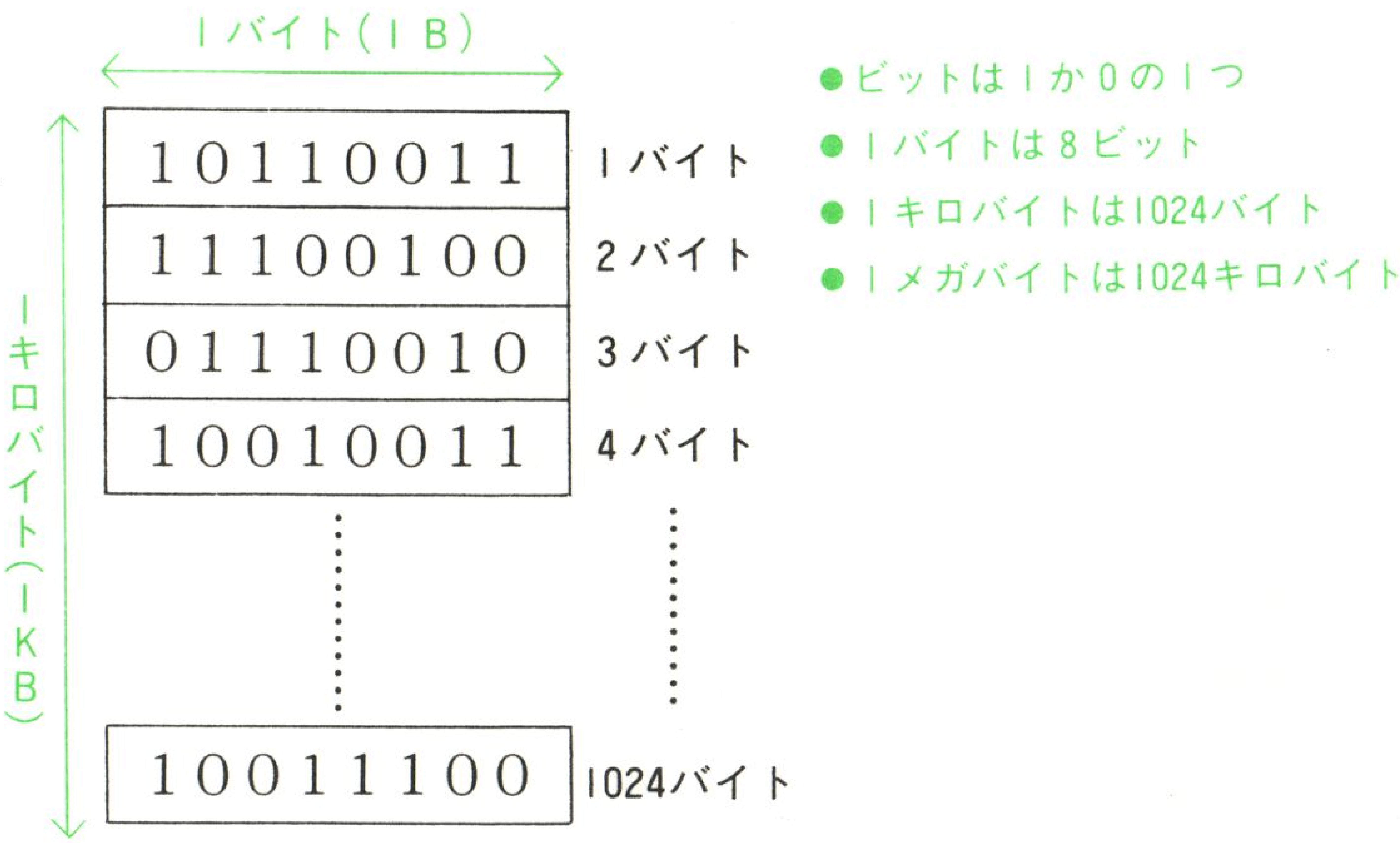
バイトあるいはキロバイトを取りあげるには、ビットから取りあげなければなりません。

さて、パソコンのなかを流れているのは、いうまでもなく電気信号ですが、電気信号そのままでは、書き表わすのに困ります。そこで、信号線に電気信号が流れているときを「1」、信号線に電気が流れていないときを「0」で書き表わすことにしています。

たとえば、信号線が四本あって、電気信号が流れている信号線を「1」、電気信号が流れていない信号線を「0」で表わすと、上の図に示すような組合せで数を書き表わすことができます。

このように、「1」と「0」で数を書き表わすことを2進法といいます。また、2進法で書き表わした数のことを2進数といいます。

ビットとは、2進法で書き表わした数の「1」か「0」の一つ（一桁）をいいます。



1バイトと1キロバイト

0 1 } 1か0の一つ（一桁）をビットという

したがって、「1」あるいは「0」が4個並んでいる場合は4ビット、8個並んでいる場合は8ビットということになります。

この「1」あるいは「0」が8個並んだ8ビットを一つの単位として、1バイト（1B）といいます。これは12個を一つの単位として、1ダースという表わし方をするのとおなじです。

そして、1バイトが1024個集まった1024バイトのことを、1キロバイト（1KB）といいます。

また、1キロバイトが1024個集まった1024キロバイトのことを、1メガバイト（1MB）といいます。

では、16ビット・パソコンがどのくらいの記憶容量を持っているかというと、現在市販されている16ビット・パソコンの場合、最低で256キロバイトの記憶容量を持っています。

ちなみに8ビット・パソコンの記憶容量は、最高で64キロバイトです。

16ビット・パソコンが最低でも256キロバイトという大きな記憶容量を持つことができ、8ビット・パソコンが最高で64キロバイトの記憶容量しか持つことができないのは、CPUのメモリ空間によります。

⑤ メモリ空間

メモリ空間とは、CPUが直接指示することのできる記憶回路の大きさをいいます。

記憶回路はCPUの指示にしたがって、記憶場所に情報を記録したり、また情報を記憶場所から送り出したりしますが、CPUのこの指示が直接届く範囲を、メモリ空間というわけです。

16ビット・パソコンのCPUの場合、メモリ空間は1メガバイト（1MB）以上となっているので、大きな記憶容量を持つことができるのです。

これに対して、8ビット・パソコンのCPUの場合は、メモリ空間が64キロバイトなので、それだけの記憶容量しか持てないのです。

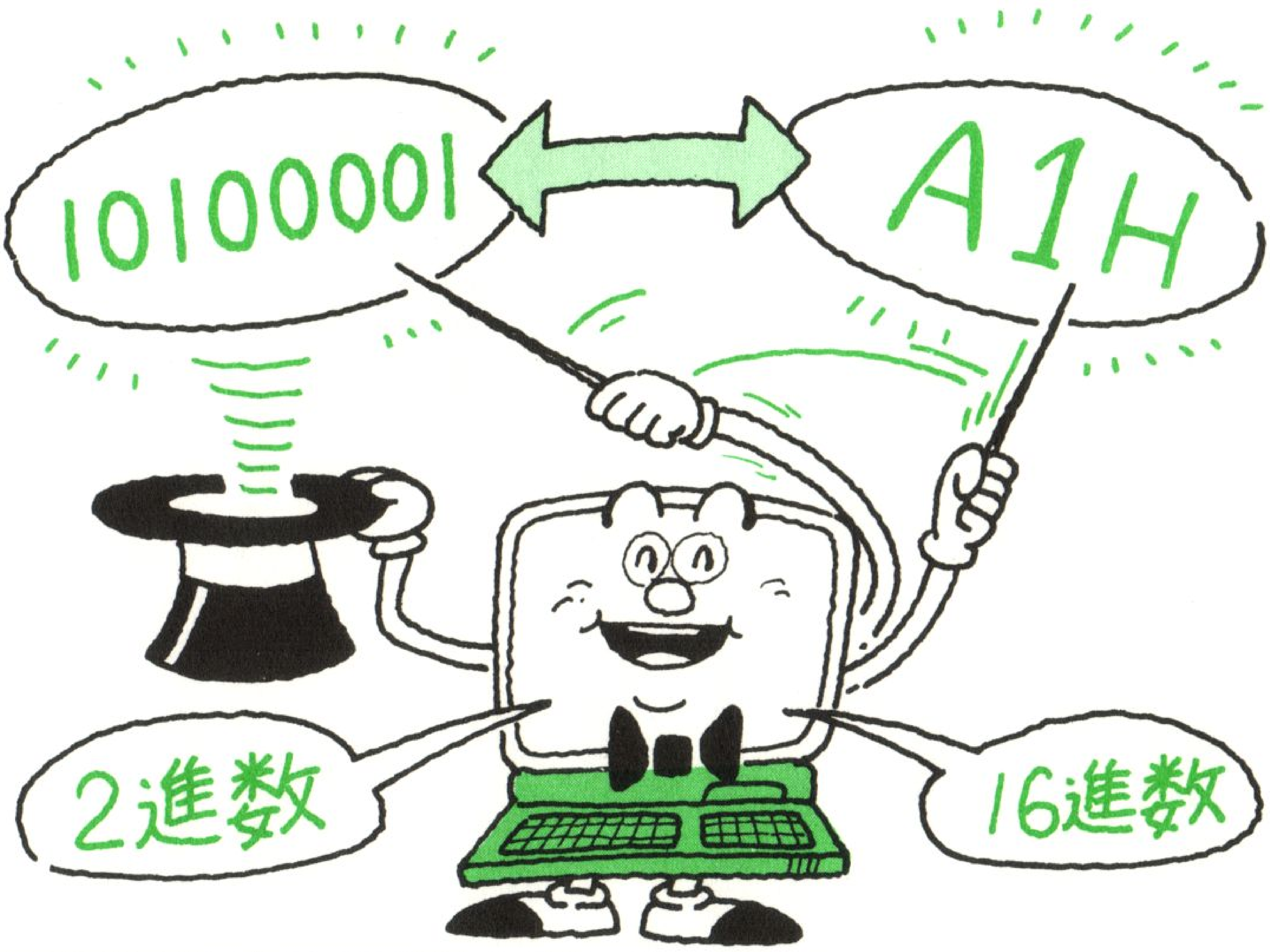
⑥ 記憶回路はROMとRAMでできている

ところで記憶回路（メモリ）は、ROMとRAMの二種類でできています。

ROM……ROM（リード・オンリー・メモリ）は、情報を書き込むことができない読み出し専用の記憶回路で、一度プログラムを書き込んでパソコンに組み込んでしまうと、電源を切っても内容が失われず、また間違ってプログラムなどを書き込もうとしてもまったく受けつけないという性質を持っています。

このような性質から、ROMには破壊されたり、消えてしまっては困るモニタ・プログラム（57頁参照）やベーシック・インタプリンタ（58頁参照）が記憶されています。

RAM……これに対して、RAM（ランダム・アクセス・メモリ）は、ROMと違って、情報を書き込んだり、書き込んだ情報を読み出したりすることができる記憶回路で、私たちが入力したプログラムやデータを記憶させる記憶回路として使われています。



16ビット・パソコンと2進法、16進法

私たちが普段、数を表わすのに使っているのは10進法です。10進法は周知のように、０１２３４５６７８９の10個の数字を使って、数を表わす方法です。

このように、私たちが10進法で数を表わすようになったのは、私たちに10本の指があるからだそうです。

2進法とは

これに対して、16ビット・パソコンにかぎらず、コンピュータのすべては、数を表わすのに2進法を使っています。

コンピュータが数を表わすのに2進法を使うの

2進数	10進数	2進数	10進数
00000000	0	00010000	16
00000001	1	00010001	17
00000010	2	00010010	18
00000011	3	00010011	19
00000100	4	00010100	20
00000101	5	00010101	21
00000110	6	00010110	22
00000111	7	⋮	⋮
00001000	8		
00001001	9		
00001010	10	11111010	250
00001011	11	11111011	251
00001100	12	11111100	252
00001101	13	11111101	253
00001110	14	11111110	254
00001111	15	11111111	255

8ビットの2進数と10進数

は、43頁で示したように、信号線に電気信号が流れていない(0)、信号線に電気信号が流れている(1)の二通りしかないからです。

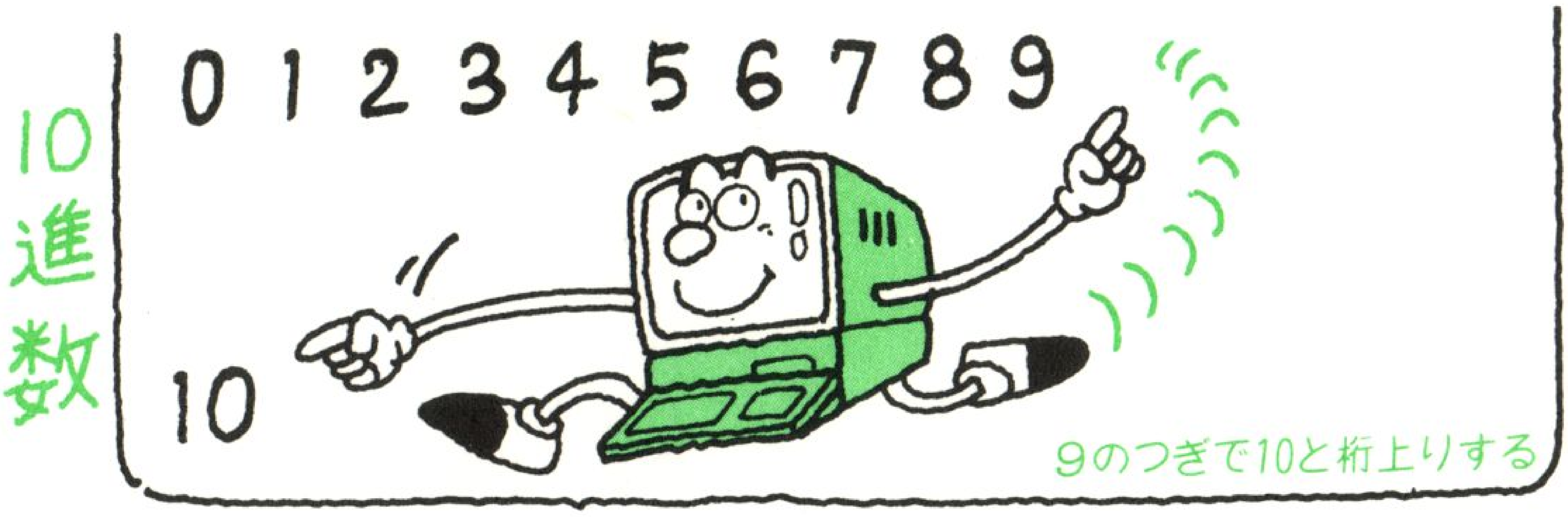
2進数では、1と0が4個並ぶ4ビットの場合、43頁で示したように、0000(10進数で0)から1111(10進数で15)までの数を表わすことができます。

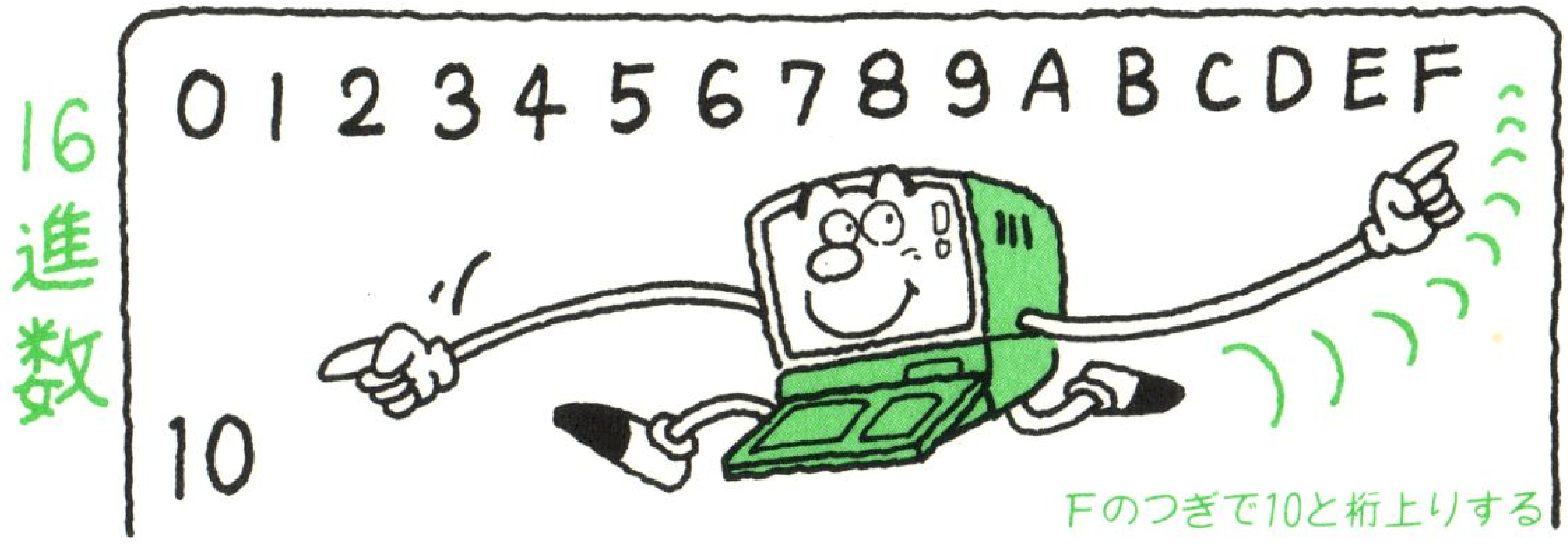
1と0が8個並ぶ8ビットの場合、上に示すように00000000(10進数で0)から11111111(10進数で255)までの数を表わすことができます。

また、1と0が16個並ぶ16ビットでは、0000000000000000(10進数で0)から1111111111111111(10進数で65535)までの数を表わすことができるようになります。

ところで、2進数についてはこれ以上詳しく知る必要はありません。これまで示してきたことを知っておくだけで十分でしょう。

なぜなら、1と0で数を表わす2進数は、16ビット・パソコンにとってはわかりやすいものであるかも知れませんが、私たちにとっては非常にわ





10進数と16進数

かりにくいものです。

　そこで2進数にかえて、16進数を使うようになっているからです。

　たとえば、ベーシックの場合では、キャラクタコード、ひらがな、漢字のコードなどを入力するときに16進数が使われます。

16進法とは

　16進法とは、16を一区切りにして表わす数といえます。10進法は10を一区切りにして表わす数、いいかえれば、0から9までが一桁の数で、9のつぎで10と桁上がりしますが、16進法は0からFまでが一桁の数で、Fのつぎで10と桁上がりします。

0 1 2 3 4 5 6 7 8 9 ←10進法の一桁の数
10 ← 9のつぎで10と桁上がりする
0 1 2 3 4 5 6 7 8 9 A B C D E F ←16進法の一桁の数
10 ← Fのつぎで10と桁上がりする

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)	(15)
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
(16)	(17)	(18)	(19)	(20)	(21)	(22)	(23)	(24)	(25)	(26)	(27)	(28)	(29)	(30)	(31)
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
(32)	(33)	(34)	(35)	(36)	(37)	(38)	(39)	(40)	(41)	(42)	(43)	(44)	(45)	(46)	(47)
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
(48)	(49)	(50)	(51)	(52)	(53)	(54)	(55)	(56)	(57)	(58)	(59)	(60)	(61)	(62)	(63)
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
(64)	(65)	(66)	(67)	(68)	(69)	(70)	(71)	(72)	(73)	(74)	(75)	(76)	(77)	(78)	(79)

16進数と10進数

16進数の9のつぎのAからFまでは、つぎのように10進数の10から15までに対応しています。

10	←	A
11	←	B
12	←	C
13	←	D
14	←	E
15	←	F
↑10進数		↑16進数

したがって、16進数の場合、Fのつぎの「10」は文字通りの「10」ではなくて、10進数の「16」ということになります。

16進法で表わす数（16進数）と10進法で表わす数（10進数）が、どのように対応しているか、上にまとめて示します。カッコのなかの数字は10進数です。

16進数にはHをつける

ところで、上に示した16進数をみた場合、AやFなどがついている数字は16進数であることがすぐわかりますが、AやFなどがついていない数字は、16進数なのか10進数なのか、まったく区別がつきません。

たとえば、ただ「30」と書かれている場合、16

進数なのか10進数なのか、判断できません。

30 ←16進数なのか10進数なのか見わけがつかない

この「30」が10進数なら、その数は文字通り30ですが、もしこの「30」が16進数だったとしたら、その数は10進数で48ですから、えらく違った数になってしまいます。

そこでこのような間違いがおきないように、16進数の終わりには、ヘキサデシマル（16進数）の頭文字Hをつけて、

30H ←16進数にはHをつける

10進数と区別することにしています。

■16進数を10進数に変換するには

まえの頁に16進数の一部を10進数に変換して示しましたが、16進数のすべての数を10進数に変換して示すことは、頁数の関係でとてもできません。したがって、書き表わされている16進数が10進数でいくつなのかを知りたいときは、16進数を10進数に変換しなければなりません。

0からFまでの一桁の16進数は、10進数でいくつなのかは比較的簡単におぼえることができるので、ここでは二桁の16進数を10進数に変換する方法を示すことにします。

たとえば、二桁の16進数A1Hを10進数に変換するとすれば、つぎのようにAと1のふたつにわけて、一桁の16進数にします。

A　1 ←A1Hをふたつにわける

そして、一桁の16進数Aと1に対応する10進数を、これにあてはめます。一桁の16進数Aは、10進数で10ですから10をあてはめ、一桁の16進数1は、10進数で1ですから1をあてはめます。

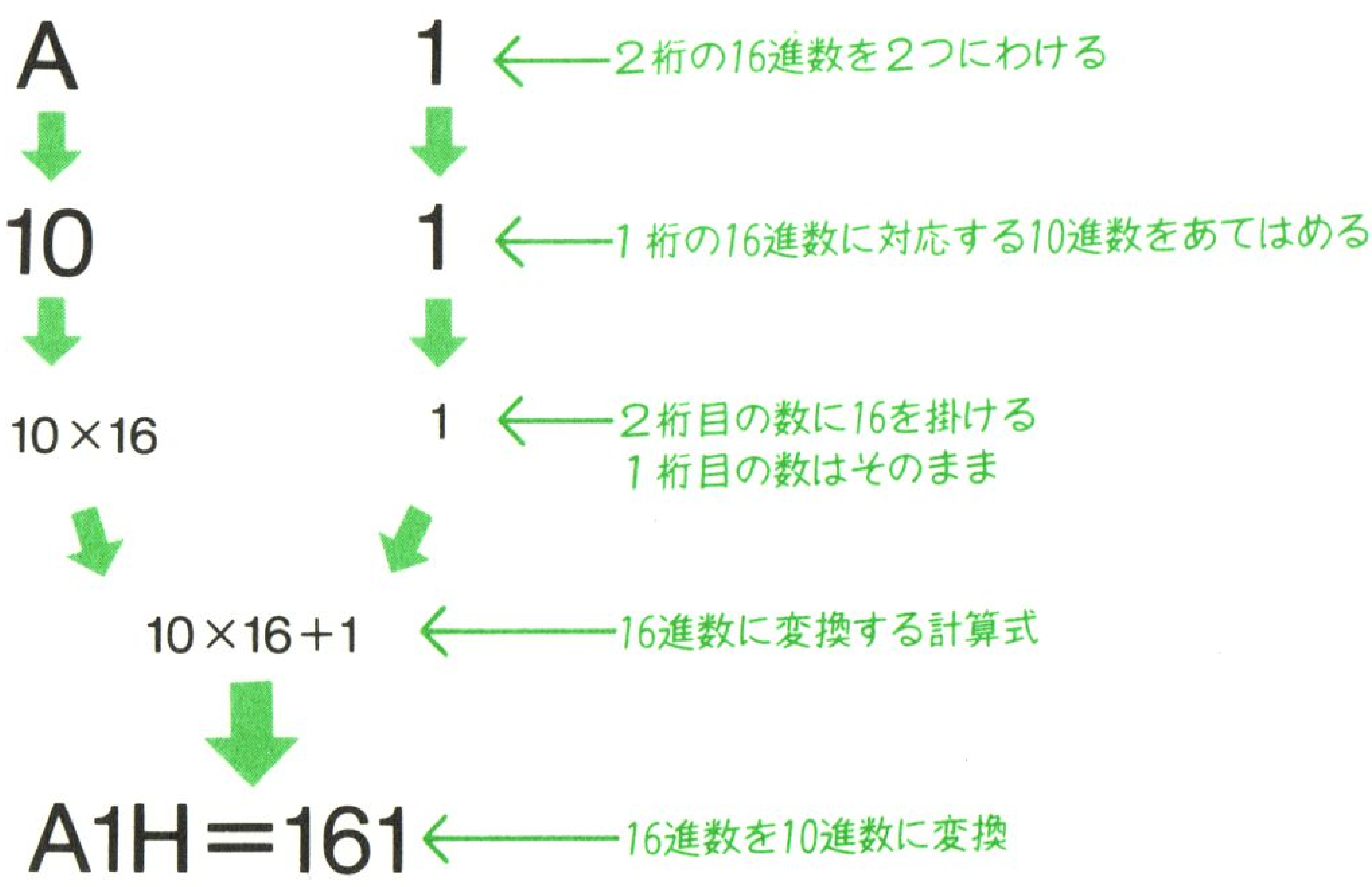


このように10進数をあてはめたら、二桁目にある数に16を掛けます。

二桁目にある数×16

この場合、二桁目にある数はAで、Aは10ですから、10に16を掛けます。

10×16



16進数を10進数に変換する方法(1)

一桁目の数はそのまま、16を掛けた二桁目の数に足します。この場合、一桁目の数は1ですから、つぎのようになります。

10×16＋1↑一桁目の数はそのまま足す

これを計算すると161ですから、Ａ１Ｈは、10進数で161ということになります。

A1H↑161（10進数）

ところで、16進数を10進数に変換するのに、このように計算していたのでは桁数が大きくなった場合に面倒です。そこで、16進数を10進数に変換する簡単な方法を示しておきます。

パソコンに電源を入れ、ＯＫとカーソルが表示されたらダイレクトモードで、つぎのように入力します（次頁の実行結果参照）。

PRINT　&H変換する16進数

画面に表示させるＰＲＩＮＴ命令のあとに、&Ｈをつけて10進数に変換する16進数を指定します。

たとえば、Ａ１Ｈを10進数に変換するとすれば、

PRINT　&HA1↑A1Hを10進数に変換

&ＨのあとにＡ１と指定します。これを入力し

```
PRINT & Hx
```
← xのところに10進数に変換する16進数を指定する

実行結果

```
print &ha1      ← A1Hを10進数に変換
 161            ← A1Hは10進数で161
Ok
print &h38ef    ← 38EFHを10進数に変換
 14575          ← 38EFHは10進数で14575
Ok
```

16進数を10進数に変換する方法(2)

て⏎キーを叩くと、A1Hが10進数の161に変換されて表示されます。

また、38EFHを10進数に変換するときは、

PRINT &H38EF

&Hのあとに38EFと指定します。これを入力して⏎キーを叩くと、38EFHが10進数の14575に変換されて表示されます。

上に実行結果を示します。

■10進数を16進数に変換するには

つぎに、10進数を16進数に変換する方法も示しておくことにします。

10進数を16進数に変換するには、つぎのようにHEX$関数を使います。HEX$は10進数を16進数に変換する関数です。

PRINT HEX$(X) ←10進数を16進数に変換

HEX$関数のカッコのなかのXに、16進数に変換する10進数を指定します。

たとえば、14575を16進数に変換するとすれば、次のようになります。

PRINT HEX$(14575)

```
PRINT HEX$(x)
```
← カッコのなかのXのところに16進数に変換する10進数を指定する

実行結果

```
print hex$(14575)
38EF
Ok
print hex$(345)
159
Ok
```
- print hex$(14575) ← 14575を16進数に変換する
- 38EF ← 14575は16進数で38EF
- print hex$(345) ← 345を16進数に変換する
- 159 ← 345は16進数で159

10進数を16進数に変換する方法

これを入力して⏎キーを叩くと、14575は16進数の38EFに変換されて表示されます。

また、345を16進数に変換するなら、つぎのようになります。

PRINT HEX$(345)

これを入力して⏎キーを叩くと、345は16進数の159に変換されて表示されます。

この実行結果も、上に示します。

4ビットの2進数は16進数の一桁で表わされる

ところで48頁で、2進数は使わないと述べました。たしかにベーシックばかりでなく、マシン語の場合でも、2進数をそのまま使うことはありませんが、2進数と16進数がどのように対応しているか、ということは知っておかなければならない事柄なので、ここで2進数と16進数の対応の仕方について取りあげます。

2進数と16進数は、つぎの頁の対応表に示すように、4ビットの2進数で表わすことができる、

2進数	16進数	10進数	2進数	16進数	10進数
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

2進数・16進数対応表

0000から1111までは、16進数の一桁で表わすことができるようになっています。たとえば、4ビットの2進数0000は16進数で0H、1010はAH、1111はFHで対応していますから、16進数は、2進数の4桁分（4ビット）をひとまとめにして、一桁の数字で書き表わすことができるようにしたものといえます。

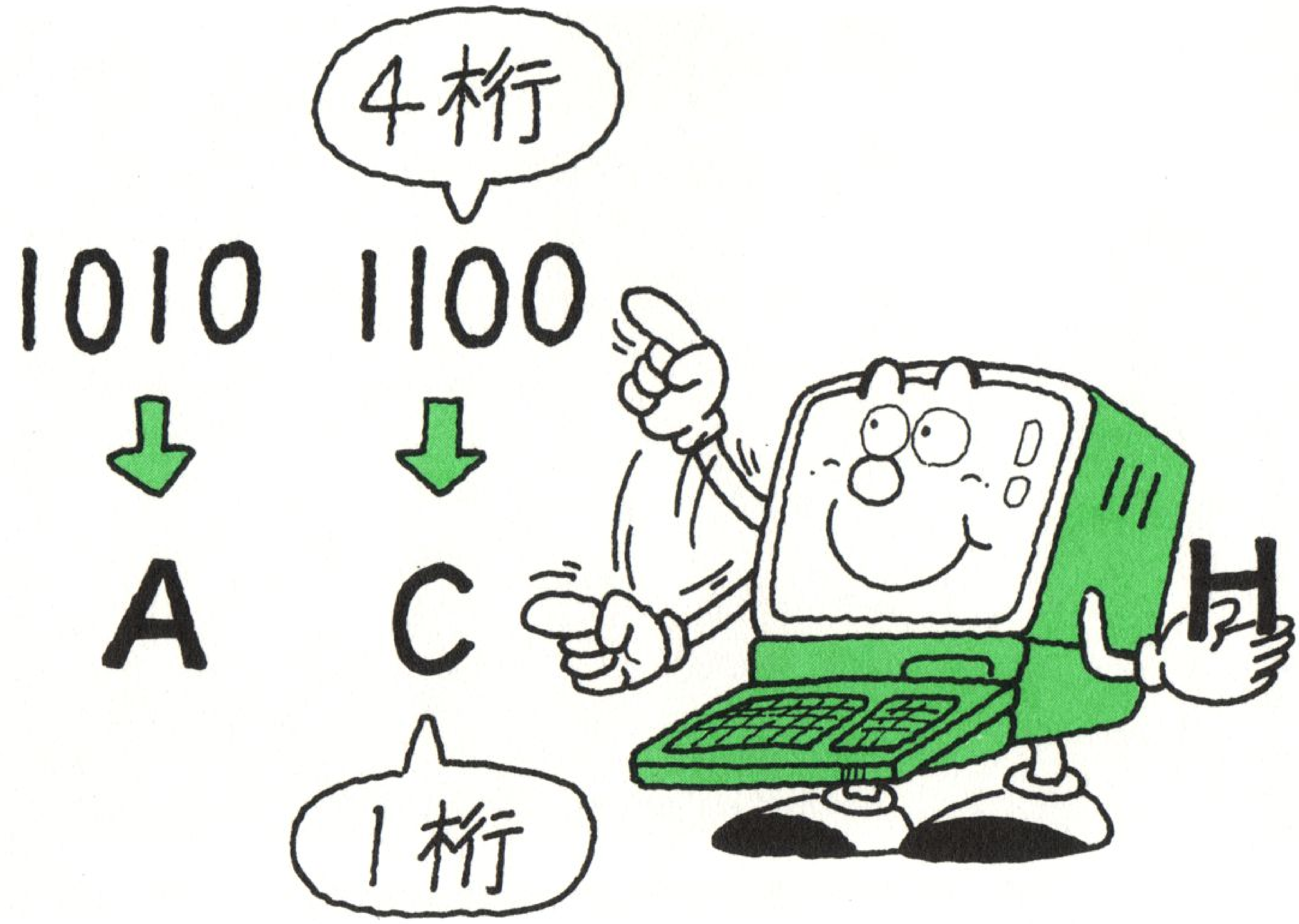
したがって、8ビットの2進数というように、桁数が4ビットの倍になっな場合でも、簡単に16進数に変換することができます。

たとえば、8ビットの2進数10101100を16進数に変換するとすれば、4ビットずつに分けます。

1010　1100

そして、4ビットの2進数1010と1100に対応する16進数を、上に示した対応表のなかから求めます。

4ビットの2進数1010は16進数でA、1100は16進数でCですから、それをつぎのようにあてはめます。



16進数は2進数の4桁を1桁で表わすようにしたもの

1010 → A
1100 → C

あとはAとCをまとめてACHとすれば、16進数に変換できます。また、ACHが10進数でいくつであるかは、52頁に示した方法で10進数に変換すればよいのです。このように8ビットの2進数の場合は、二桁の16進数に変換されます。

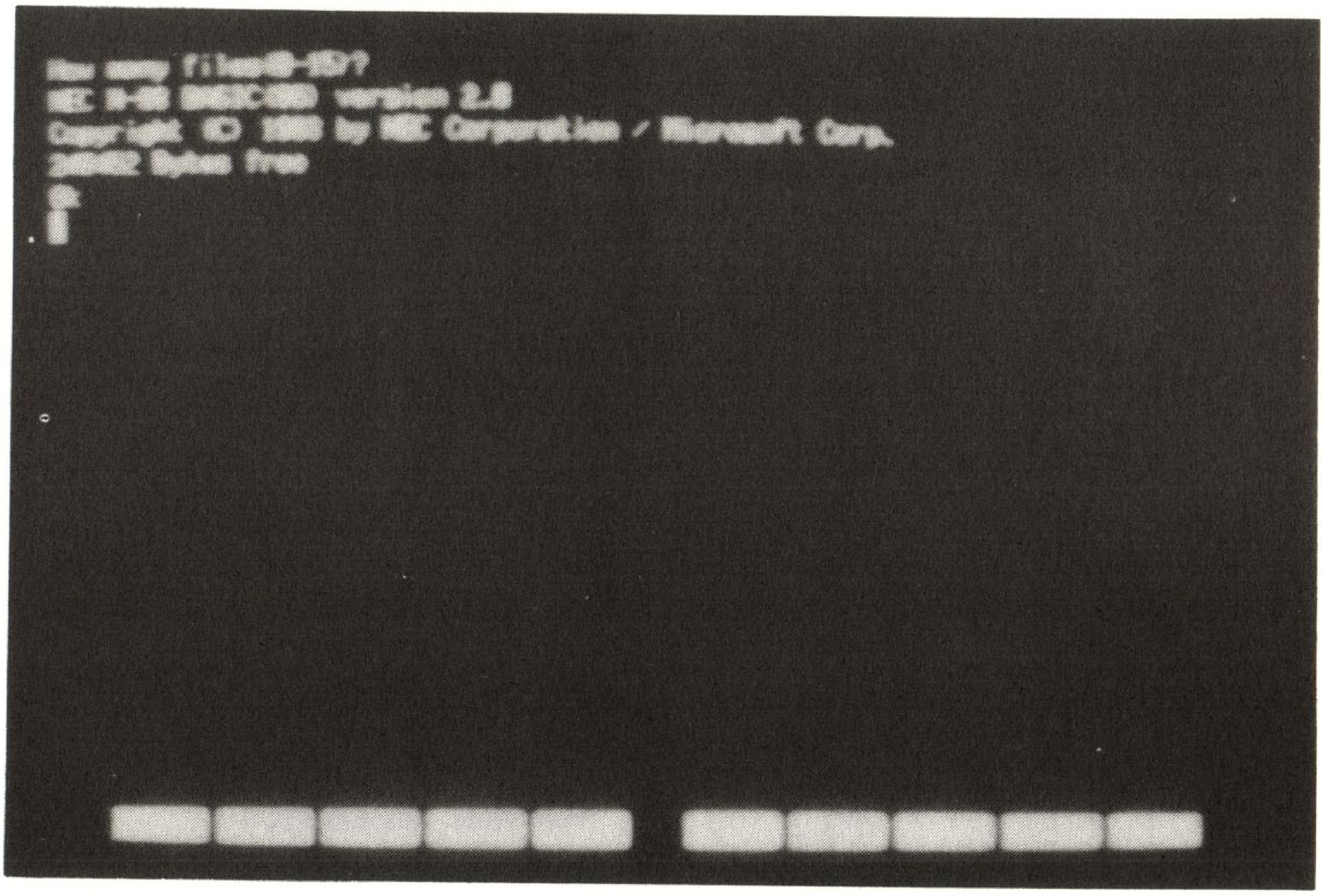
16ビットの2進数を16進数に変換する場合も4ビットずつに分け、一桁の16進数をあてはめます。

0011 1000 1010 0001 ←16ビットの2進数
↓
0011　1000　1010　0001 ←4ビットずつにわける
↓
3　8　A　1 ←1桁の16進数をあてはめる

あとはそれをまとめればよいのですから、簡単に16進数に変換することができます。

0011100010100001 → 38A1H

16ビットの2進数の場合は、このように4桁の16進数に変換されます。

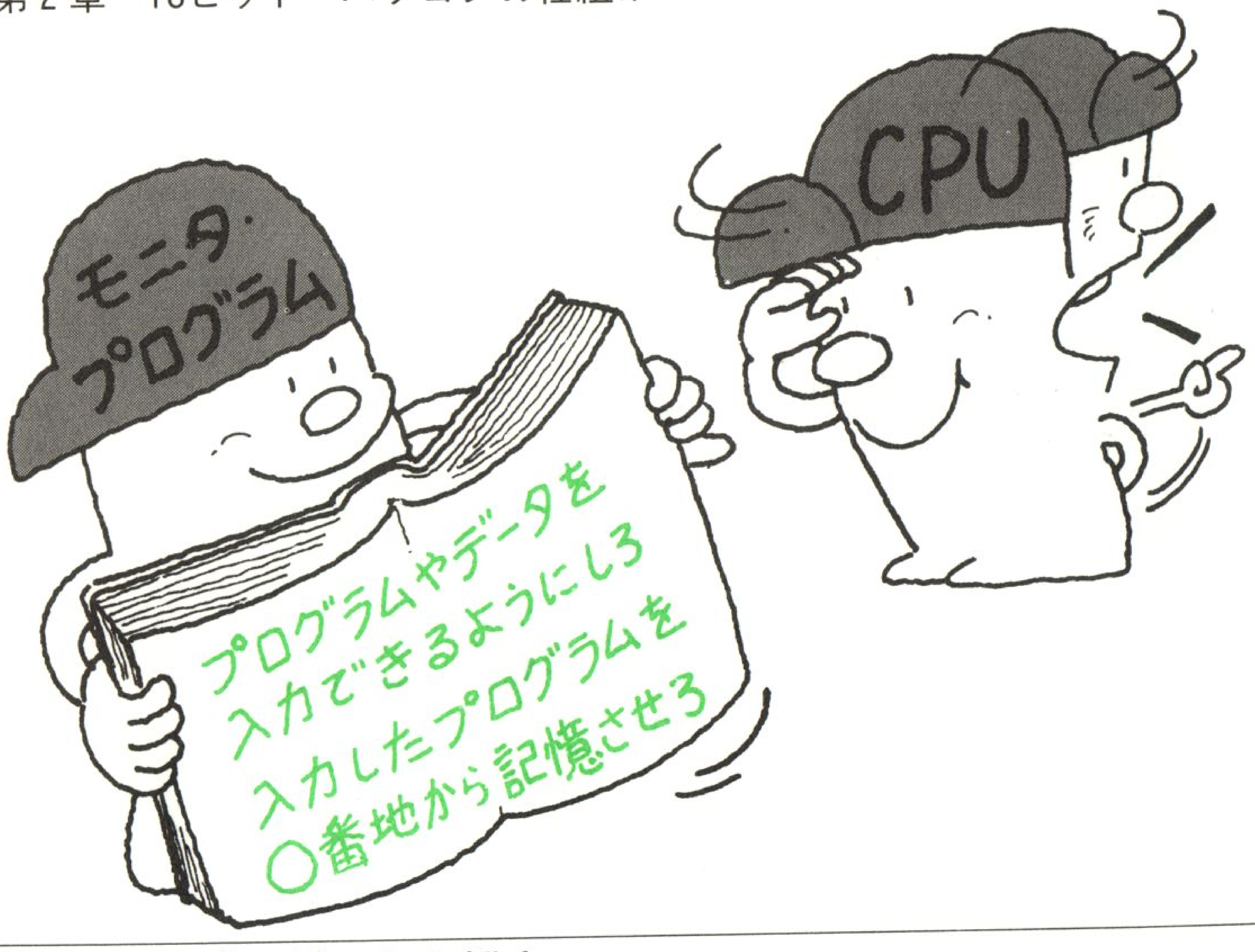


電源を入れるとメッセージが表示される

モニタ・プログラムとベーシック・インタプリタ

コンピュータは、プログラムがないとなにもすることができません。私たちが、パソコンを作動させるために電源を入れても、ただパソコンのなかに電気が流れるだけです。電気が流れるだけでは、パソコンに仕事をさせるためのプログラムやデータの入力さえもできません。

ところがパソコンに電源を入れると、上の写真に示すようにメッセージが表示されて、ベーシックのプログラムやデータを入力することができるようになるばかりでなく、入力したプログラムやデータは電気信号に変換されて、記憶回路のなかの決められた記憶場所から、順に記憶されていきます。またプログラムを実行させると、記憶場所から順にプログラムが取り出されていきます。



モニタ・プログラムの働き

モニタ・プログラムとは

このようなことが自動的に行われるのは、電源を入れるとパソコンの基本的な機能を働かせるための手順を示したプログラムが、あらかじめＲＯＭ（リード・オンリー・メモリ）という記憶回路のなかに記憶されていて、私たちがパソコンに電源を入れると同時に、このプログラムに書かれている手順をＣＰＵが読み取って、パソコンの全体に制御信号を出し、プログラムに示された手順通りに基本的な機能を働かせるようになっているからです。

このような基本的な機能を働かせるためのプログラムを、モニタ・プログラムといいます。

モニタ・プログラムによるパソコンの働きはいろいろです。パソコンの働きのなかで、私たちが入力したプログラムによる働き以外のいろいろな働きは、すべてこのモニタ・プログラムによるものなのです。



ベーシックは内部言語で記憶される

ベーシック・インタプリタとは

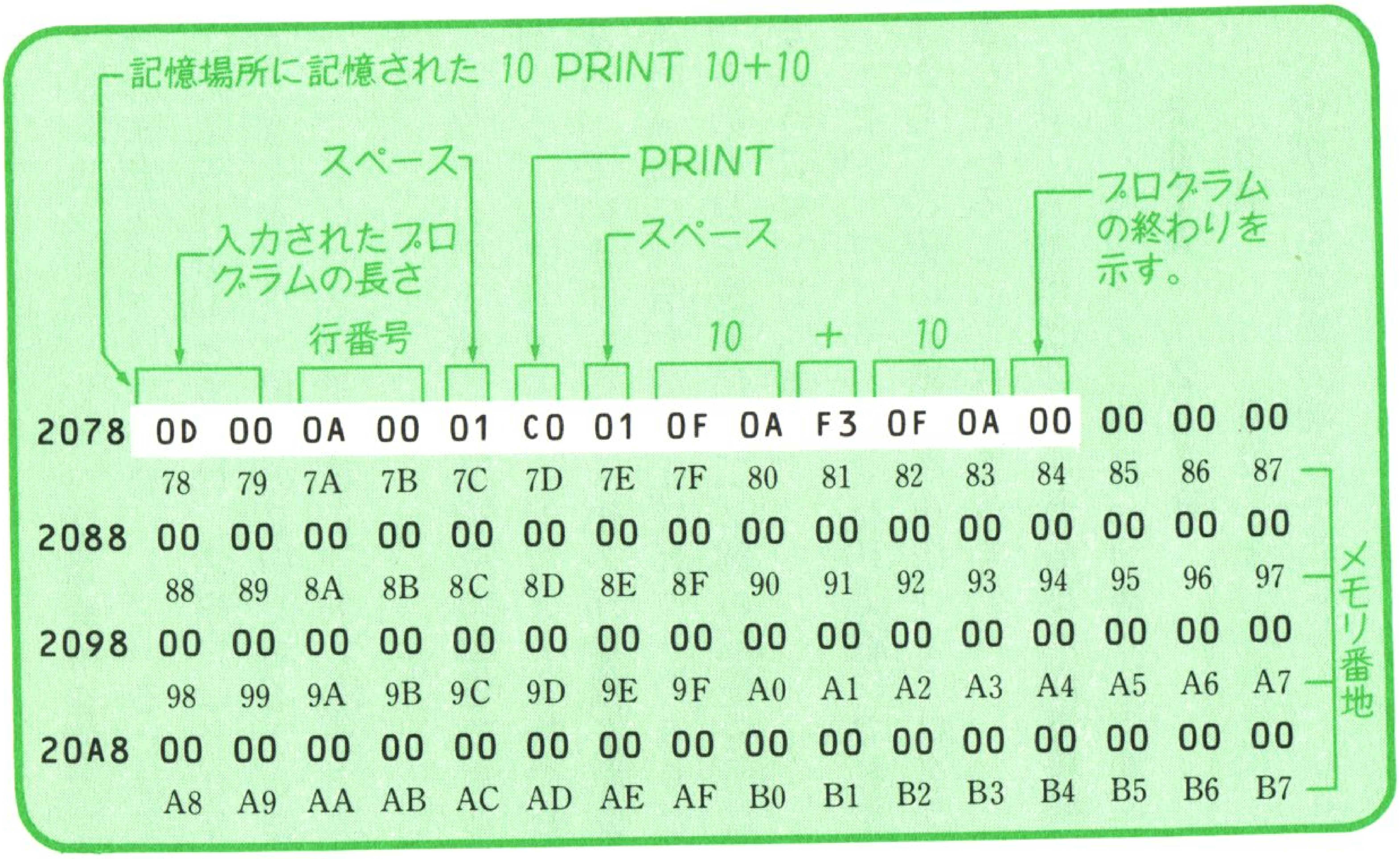
さて、私たちがベーシックのプログラムやデータを入力すると、入力されたプログラムやデータは電気信号に変換されて、決められた記憶場所から順に記憶されていくわけですが、ベーシックのプログラムやデータの場合は、電気信号といっても、そのままCPUが解読できる電気信号ではありません。

ベーシックのプログラムやデータは、内部言語あるいは中間言語と呼ばれる特殊な電気信号に変換されて、記憶場所に順に記憶されるようになっています。

では参考までに、ベーシックのプログラムやデータが、どんな電気信号に変換されて、記憶場所に順に記憶されるかを示します。

■ベーシックの記憶のされ方

ここでは、つぎの簡単なプログラムを入力することにします。



10 PRINT 10+10の記憶のされ方

10 PRINT 10＋10

最初の10は、行番号です（143頁参照）。PRINT（プリント）は、「ディスプレイに表示せよ」というベーシックの命令です。10＋10はあらためて述べるまでもないでしょう。

つまり、このプログラムは、「10＋10の足し算を行って、その結果をディスプレイに表示せよ」というものです。

このプログラムを、キーボードのキーを叩いて入力すると、上に示すような電気信号に変換されて、２０７８番地の記憶場所から２０８４番地までの記憶場所に順に記憶されていきます。

記憶回路のなかを直接画面に表示させた場合、記憶場所に記憶されている電気信号は、すべて16進数で表示されます。

さて、２０７８番地と２０７９番地の記憶場所に記憶されている０Ｄと００は、入力されたプログラムの長さを示しています。

2078　2079
0D　00
⬆プログラムの長さ

いいかえれば、いくつ記憶場所を使ってプログラムを記憶しているかを示しているわけです。

ところで０Ｄと００は、実際には０００Ｄとなります。そして、最初の２桁である００を上位ビット、あとの２桁である０Ｄを下位ビットといいます。

パソコンの１個の記憶場所に記憶できるのは８ビット、つまり００・０Ｄなどの２桁の16進数と決まっていますから、４桁の16進数を記憶する場合はふたつに分けて、上位ビットの００と下位ビットの０Ｄとを逆にして２個の記憶場所に記憶するようになっています。

したがって、０００Ｄを10進数になおすと13ですから、13個の記憶場所を使ってプログラムを記憶していることを示しています。

２０７Ａ番地と２０７Ｂ番地の記憶場所に記憶されている０Ａと００は、行番号10です。

207A　207B
0A　00 ↑行番号10

この場合もさきの場合とおなじで、０００Ａです。０００Ａは10進数で10です。

２０７Ｃ番地の記憶場所に記憶されている０１は、行番号10とＰＲＩＮＴの間をあけるために、スペースキーを叩いて入力した空白です。

207C　01 ↑空白

２０７Ｄ番地の記憶場所に記憶されているＣ０は、ＰＲＩＮＴです。

207D　C0 ↑PRINT

２０７Ｅ番地の記憶場所に記憶されている０１は、ＰＲＩＮＴと10＋10の間をあけるために、スペースキーを叩いて入力した空白です。

207E　01 ↑空白

７０７Ｆ番地と２０８０番地の記憶場所に記憶されている０Ｆと０Ａは、10＋10の最初の10です。

207F　2080
0F　0A ↑最初の10

この場合は０Ａ０Ｆではなく、０Ａが10です。０Ｆは０Ａが整数であることを示す記号です。

２０８１番地の記憶場所に記憶されているＦ３は、＋記号です。



ベーシック・インタプリタはベーシック・マシン語辞典

2081　F3　⬆　＋の記号

２０８２番地と２０８３番地の記憶場所に記憶されている０Ｆと０Ａは10＋10の終りの10です。

2082　2083
0F　0A　⬆　終わりの10

２０８４番地の記憶場所に記憶されている００は、プログラムの終りを示します。

2084　00　⬆　プログラムの終わり

このように、ベーシックのプログラムやデータを入力すると、内部言語という電気信号に変換されて記憶場所に記憶されるわけですが、これはあくまでもベーシックですから、ＣＰＵには解読することができません。

そこで、ＣＰＵがベーシックの命令を解読することができるように、ベーシックの命令のひとつひとつを、ＣＰＵが解読できる言葉で書き表わした辞書が、ＲＯＭという記憶回路のなかにあります。

この辞書のことを、ベーシック・インタプリタといいます。



CPUはベーシック・インタプリタにあるマシン語を実行する

ベーシック・インタプリタは ベーシック・マシン語辞典

ＣＰＵが解読することができる言葉は、マシン語だけです。

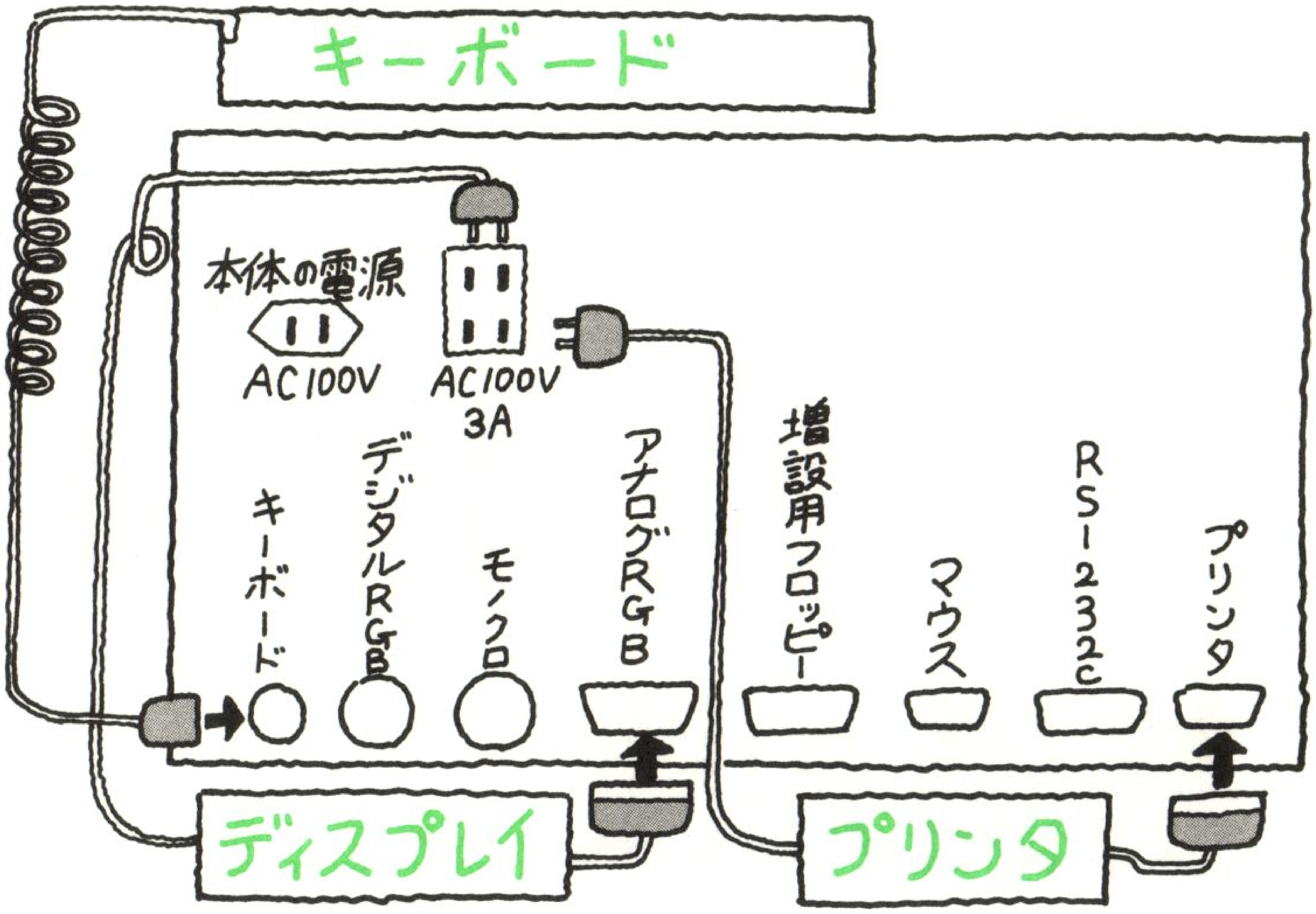
したがって、ベーシック・インタプリタは、ベーシックの命令のひとつひとつを、マシン語で書き表わしています。ちょうど英和辞典とか、和英辞典のようにです。

ＣＰＵは、プログラムなどベーシックの命令を読み込むたびに読み込んだベーシックの命令の項を引いて、マシン語で示されていることを実行していきます。

16ビット・パソコンのＣＰＵの処理スピードは高速なので、あたかもＣＰＵが、直接ベーシックのプログラムを実行しているようにみえますが、実際にはこのように、ひとつひとつの命令をベーシック・インタプリタという辞書を参照し、命令の意味を理解しながら、ベーシックのプログラムを実行しているわけです。

第3章

パソコン操作の基礎知識



パソコン本体と周辺機器の接続

パソコン本体と周辺機器の接続

まえの章で、パソコンの仕組みについて示しました。パソコンのおおよその仕組みがわかれば、つぎはパソコンの操作です。

パソコンを操作するには、まず情報を入力するキーボード、情報を画面に表示するディスプレイ、文字を印字して打ち出すプリンタなどの周辺機器を、パソコン本体に接続します。

情報をディスクに記録したり、記録した情報を入力したりするディスク装置は、パソコン本体のボディーに組み込まれているので、接続する必要はありません。

パソコン本体と周辺機器の接続の仕方を、ＰＣ－９８０１ＶＦ２やＶＭ２を例にして示すと、上の図のようになります。

図は本体の背後からみたもので、いろいろなコネクタ（接続端子）が並んでいますが、それぞれのコネクタには、何を接続すればよいのか示されています。

キーボードとプリンタを接続するコネクタはそれぞれ一つしかないので迷うことはありませんが、ディスプレイを接続するコネクタには、モノクロ、アナログＲＧＢ、デジタルＲＧＢの3種類があります。

このうちのモノクロと書かれているコネクタには、白黒のディスプレイを接続します。

デジタルＲＧＢ、アナログＲＧＢと書かれているコネクタには、カラーディスプレイを接続しますが、カラーディスプレイが、デジタル型のものであればデジタルのコネクタに接続し、アナログ型のものであればアナログのコネクタに接続することになります。

これらのコネクタに加えて、ＰＣ－９８０１ＶＦ２やＶＭ２には、100ボルト電源の差し込みがあります。これは、ディスプレイやプリンタの電源をここから取るためです。

図に示したように、ディスプレイやプリンタの電源をここから取るようにしておくと、本体に電源を入れると同時に、ディスプレイやプリンタにも電源が入り、逆に本体の電源を切れば、ディスプレイやプリンタの電源も切れるようになります。この場合、ディスプレイやプリンタのスイッチは入れたままにしておきます。

ところで、さきに示したパソコンに接続する周辺機器のうち、キーボードとディスプレイの二つは、パソコンの操作に欠かすことのできないものですが、プリンタはなくても、パソコンを操作するという点では支障をきたしません。

ただ実際に、ワープロソフトやマルチプランなどのアプリケーションソフトを使ってパソコンで仕事を行う段階になったときは、プリンタがないと、実行結果をプリントアウトすることができませんし、なにかにつけて不自由することになるので、揃えておく必要があります。

プログラムなどを入力するためのキーボードの操作

さて、パソコンにキーボードなどの周辺機器を接続して、本体と周辺機器のスイッチを入れると、パソコンの操作ができます。

パソコンの操作で、まず、おぼえなければならないこと、また、慣れなければならないことは、キーボードの操作です。

プログラムやデータを入力するには、キーボードのうえに並んでいるキーを一つ一つ叩いて行わなければならないので、キーボードの操作ができなければ、パソコンに命令を与え、仕事をさせることはできません。

キーボードのキーには大胆に触れること

キーボードの操作をおぼえる、慣れるには、キーボードのキーに指を触れることです。

キーボードのキーは、少々のことで壊れることはありませんし、また、どんなキーを叩いても、叩いたキーの文字がディスプレイに表示されていくだけですから、ABC、イロハなど、どんな文字でもかまいませんから、とにかくキーボードのキーに大胆に指を触れることです。そのうちに、自然に指が必要とするキーに向かって動いていくようになります。

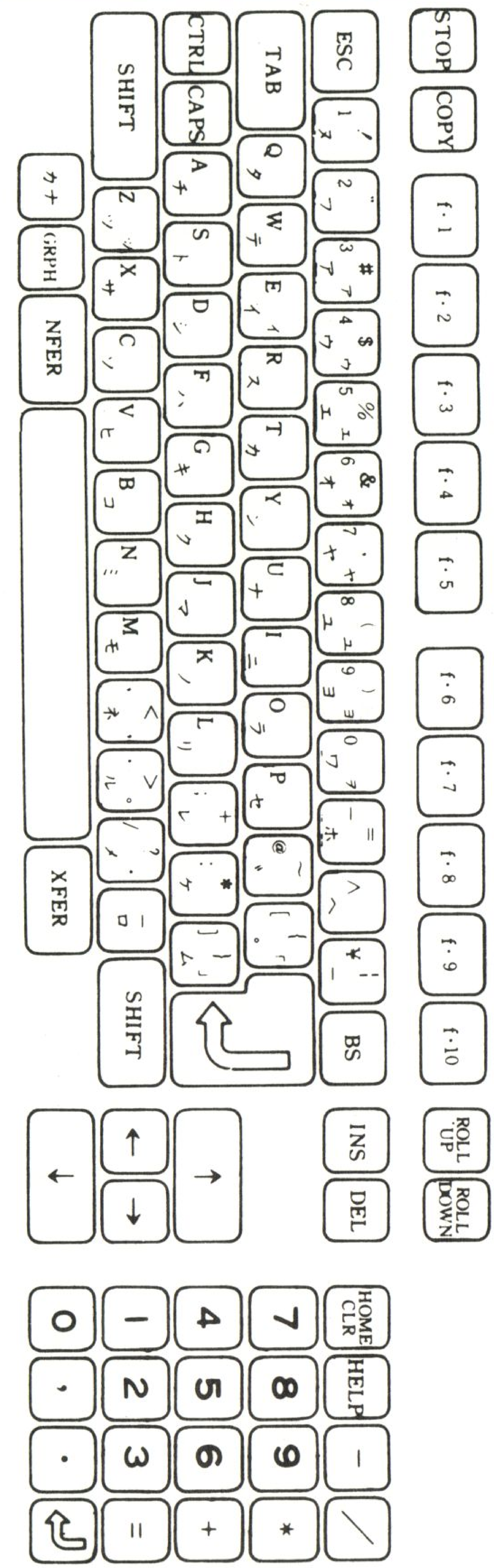
キーボードの操作とは、ただ単に、それぞれのキーの働きを知るということではなくて、指が自然に必要とするキーに向かって動いていくようになることをいうのです。

キーボードのキー配列

パソコンのキーボードのキーの配列がどのようになっているか、つぎの頁の図に示します。図に示すキーボードはPC－9801VX21のものですが、VM21のキーボードのキーの配列もまったくおなじです。

ほかの機種の場合でも、文字キーや数字キーの配列はおなじです。

PC-9801VX21のキーボードのキー配列



```
How many files(0-15)? ↵    ←── ↵キーを叩く
NEC N-88 BASIC(86) version 2.0    ←── ①を参照
Copyright (C) 1983 by NEC Corporation / Microsoft Corp.
                                        ↖── ②を参照
248452 Bytes free    ←── ③を参照
Ok    ←── ベーシックを入力してもよいという表示
■    ←── カーソル
```

①はNECのN88-BASIC(86)というベーシックで働くということを示している。
②はNECとマイクロソフト社で1983年に作成したものであることを示している。
③は248452バイトの記憶回路が自由に使えるということを示している。

スイッチを入れるとメッセージを表示する

HOME CLR キーは画面クリア

スイッチを入れると、上に示すように、ディスプレイの画面にメッセージが表示されます。PC−9801VX21やVM21の場合は、まず、

How many files(0-15)?

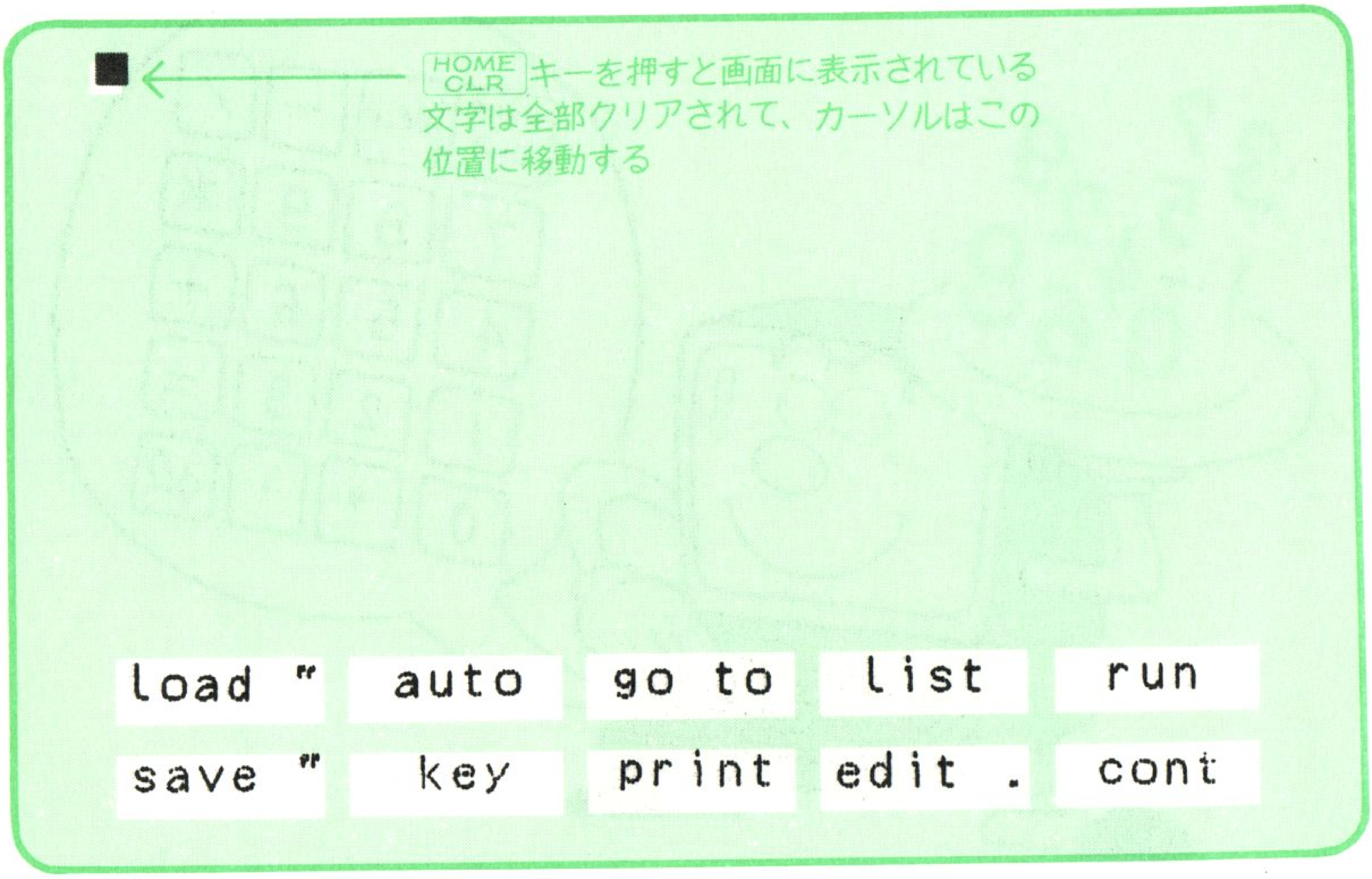
と表示されます。この意味は「いくつのファイルを使うか?」ということです。

ふつうの場合、この質問は無視してかまわないので、ここでは無視することにしますから、一般キーの右脇にある↵を叩きます。

テンキーの右下隅にある↵キーもおなじ働きのキーですから、これを叩いてもかまいません。

↵キーを叩くと、上に示すように、最初に表示されたメッセージの下に、さらに3行のメッセージとOKが表示されます。その下に四角い白いものが点滅します。

3行のメッセージは、どうということはありま



HOME CLR キーは画面に表示されている文字をクリアする

せん。まえの頁の図を参照してください。

OKとカーソルの意味

さて大切なのはOKと、点滅している四角い白いものです。

OKは、ベーシックを入力してもよいということを示しています。

また、点滅している四角い白いものはカーソルといい、私たちがキーボードのキーを叩いて入力した文字が画面に表示される場合、必ずこのカーソルが点滅している位置に表示されるようになっています。

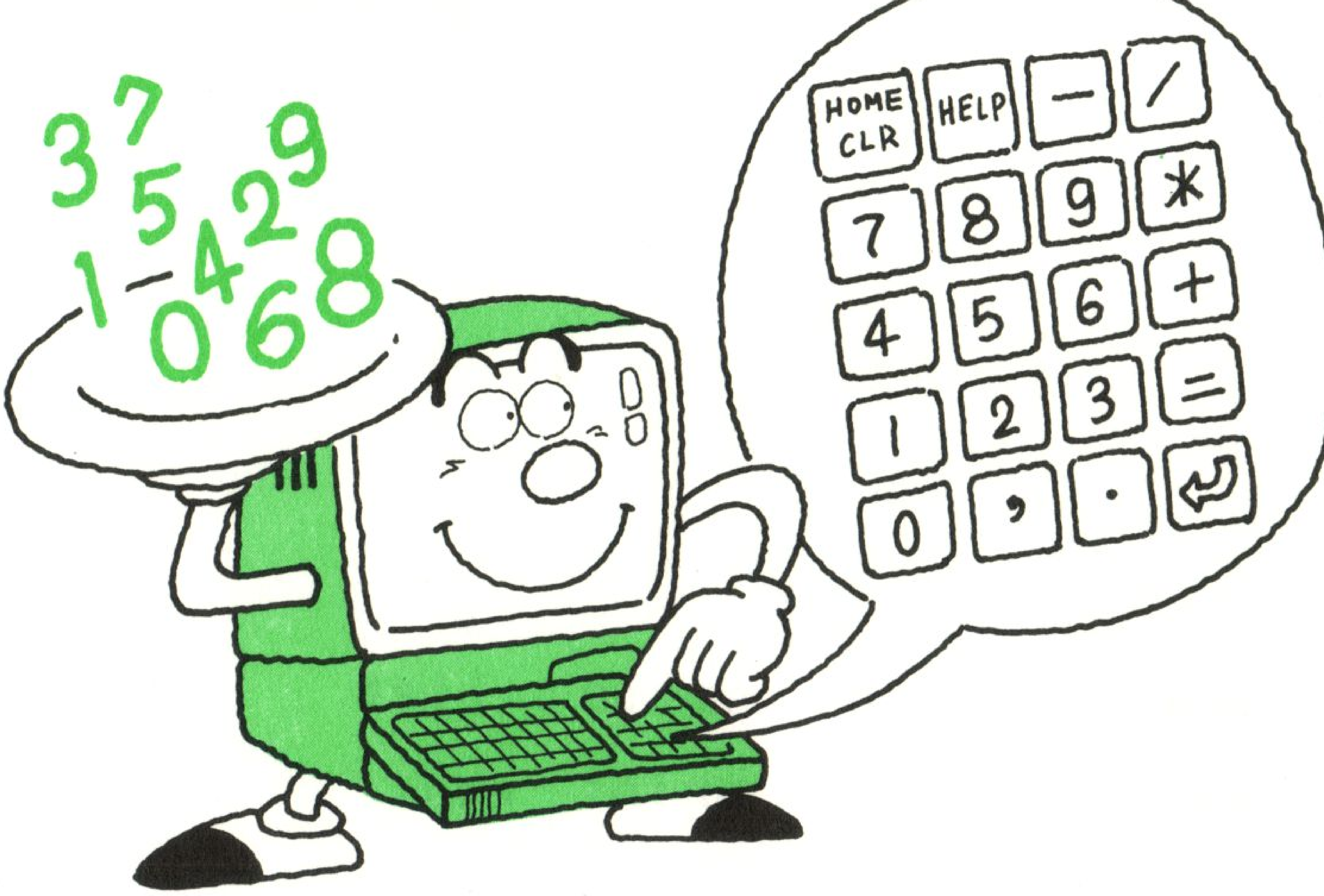
ところで、画面にメッセージが表示されたままでもベーシックを入力することができますが、画面に表示されているメッセージを全部消してから、ベーシックを入力することにします。

画面に表示されている文字を全部消すには、テンキーの左上隅にある、

HOME CLR

↑ 画面に表示されている文字を全部消す

と書かれているキーを押すと、上に示すように、



数字の入力はテンキーを使う

画面上の文字は全部消され、カーソルは画面の左上に移動して点滅します。

このように、画面に表示されている文字を全部消すことを、画面をクリアする、あるいは画面クリアといいます。

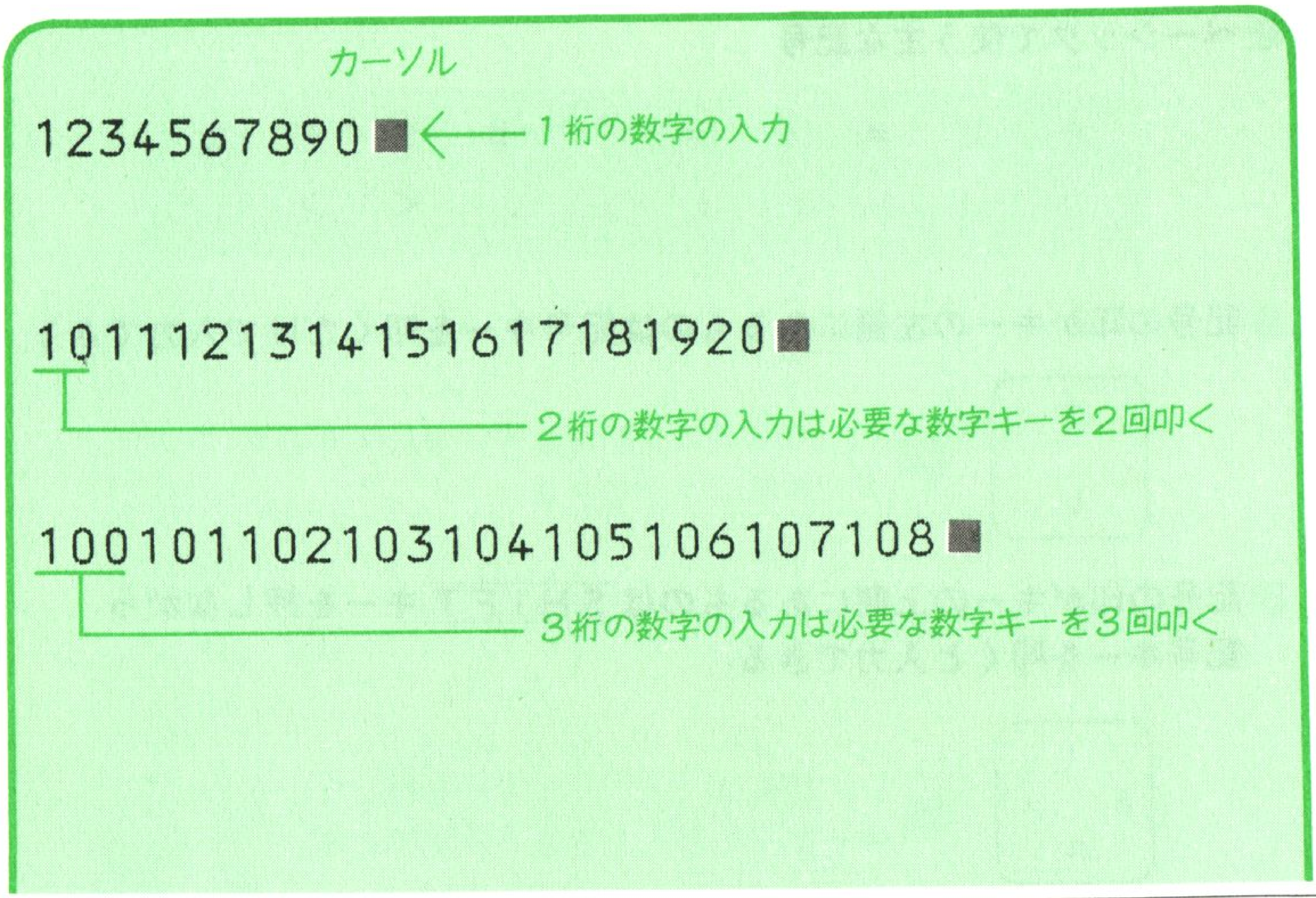
テンキーは数字の入力

画面をクリアしたら、最初はもっとも簡単な数字を入力してみます。

さて、数字キーは、一般キーの上の段とテンキーの両方にあります。

数字を入力する場合には、一般キーの上の段に並んでいる数字キーは使わないで、テンキーに並んでいる数字キーを使います。

テンキーを使う理由は、一般キーの上の段に並んでいる数字キーは叩きにくく、テンキーの数字キーのほうが叩きやすいからです。数字キーを叩きやすいということは、数字を間違いなく入力することができる、ということですから注意してく



数字キーを叩くと叩いたキーの数字が画面に表示される

ださい。

さて、テンキーにある[1]のキーを叩くと「1」が入力されて、カーソルが点滅している位置に表示されます。そして、カーソルは1文字分、右に移動して点滅します。テンキーの[2]のキーを叩くと「2」が入力されて、カーソルが点滅している位置に表示されます。そして、カーソルはまた1文字分、右に移動して点滅します。

カーソルについては89頁でも示しますが、カーソルはこのように、叩かれたキーの文字を表示する位置を示している、ということです。

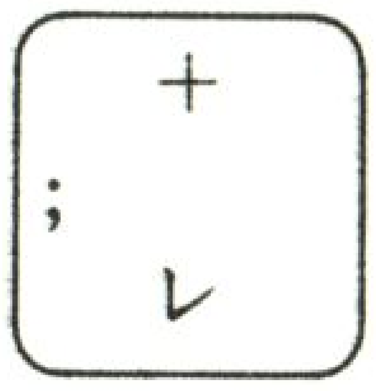
ところで「0」から「9」までの1桁の数は、[0]から[9]までの数字キーを叩けばよいわけですが、二桁の数や三桁の数の場合の入力は、二桁の数のときは必要な数字キーを2回叩き、三桁の数のときは必要な数字キーを3回叩きます。

つまり「16」という数の場合は、[1]と[6]のキーを叩き、「139」という数の場合は、[1]と[3]と[9]のキーを叩けばよいわけです。

■ベーシックで使う主な記号

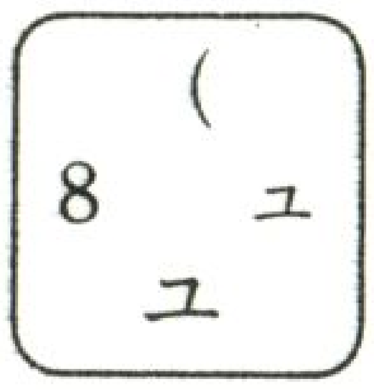
－　／　＊　＋　＝　”　＃　＄　％　＆
´　（　）　；　：　，　．　＜　＞

■記号の印がキーの左側にあるものは記号キーを叩くだけで入力できる



「；」セミコロン記号はキーを叩くだけで入力することができる

■記号の印がキーの上側にあるものは［SHIFT］キーを押しながら記号キーを叩くと入力できる



「（」記号は［SHIFT］キーを押しながらキーを叩くと入力することができる

記号キーと記号の印の位置

記号キーは記号の入力

記号の入力の仕方は、数字の入力の仕方より多少複雑です。

上にベーシックで使う主な記号を示します。

上に示した記号のうち、つぎの記号キーは、

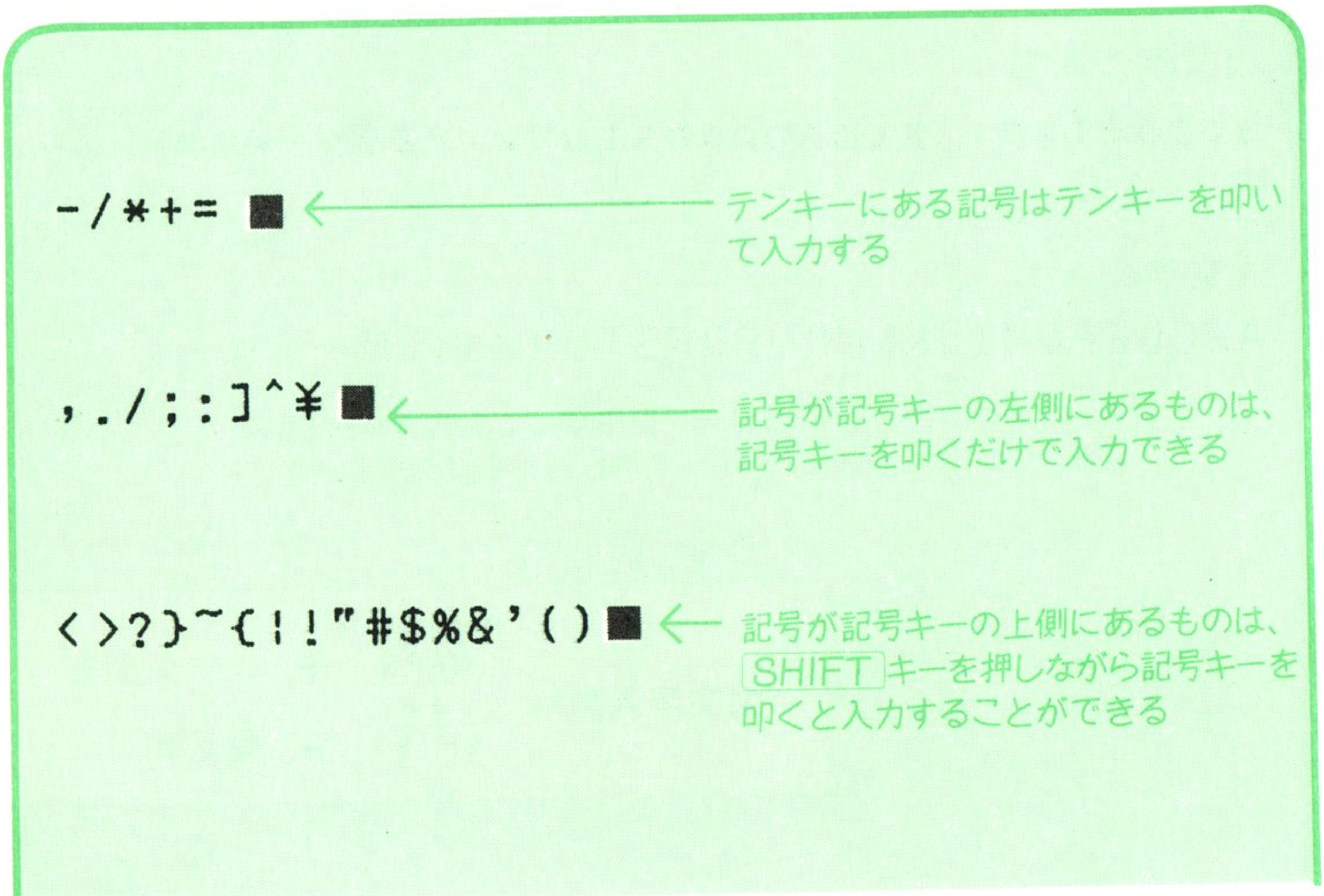
＋、－、／、＊ ←テンキーを叩いて入力

一般キーとテンキーの両方にありますが、テンキーのほうが叩きやすいので、テンキーを叩いて入力します。

そのほかの記号キーは、一般キーの一番上の段から四段目までにありますが、上の図に示すように、これらの記号キーは記号の印がキーの左側に示されているものと、キーの上側に示されているものとがあります。

記号の印がキーの左側に示されている記号は、ただキーを叩くだけで入力することができます。

たとえば、「，(コンマ)」、「・(ピリオド)」、「；(セミコロン)」、「：(コロン)」などの記号は、記



記号の入力

号の印がそれぞれのキーの左側に示されているので、キーを叩くだけで上に示すように、画面に表示されます。

これに対して、記号の印がキーの上側に示されている記号は、ただキーを叩くだけでは入力されません。SHIFT（シフト）と書かれたキーを押しながらキーを叩くと入力されます。

たとえば、「#」、「'」、「%」、「<」、「^」などの記号は、キーの上に記号の印が示されているので、SHIFTキーを押しながらキーを叩くと、上に示すように画面に表示されます。

SHIFTキーは、メインキーの下から2段目の両脇にあります。

アルファベットのキーは英文字を入力

英文字の入力は、一般キーの中央にまとめて並べられている、AからZまでのアルファベットが示されているキーを叩いて行います。

アルファベットのキーをみると、AからZを示

小文字の入力

abcdefghijklmnopqrstuvwxyz■←英文字キーを叩く

大文字の入力

ABCDEFGHIJKLMNOPQRSTUVWXYZ■←

[CAPS]キーをロックして英文字キーを叩く
[SHIFT]キーを押しながら英文字キーを叩く

大文字入力＝ キャピタルロックキー [CAPS] ← ロックする
シフトキー [SHIFT] ＋ [英文字]

英文字の入力

す文字は、すべてキーの左側に示されていますから、キーを叩くだけで入力することができます。

ただし[A]から[Z]までの文字は大文字で示されていますが、キーを叩くと叩いたキーの英文字は、上に示すように画面に小文字で表示されます。

■大文字の入力は[CAPS]キーをロック

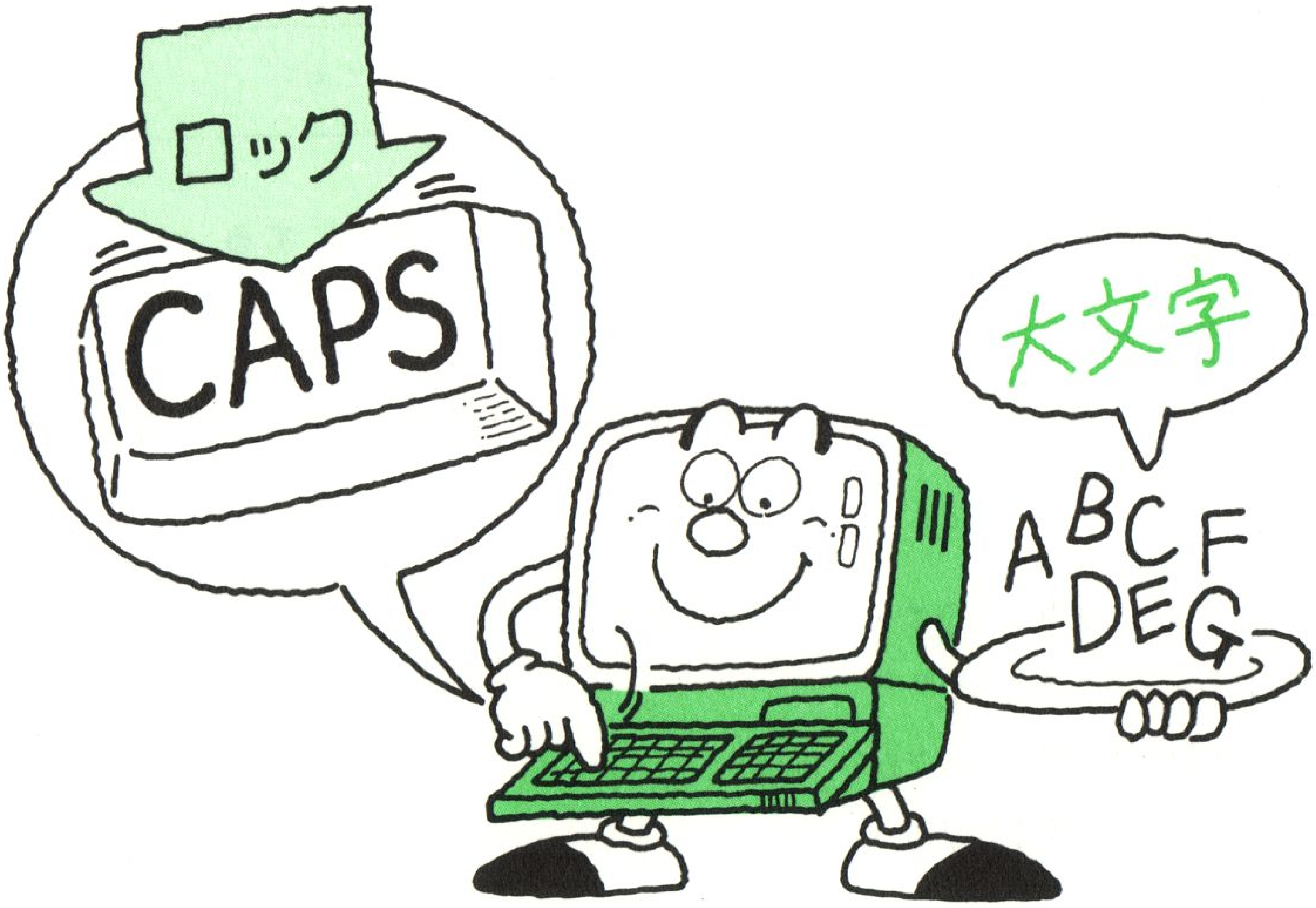
ふつうの場合、小文字で入力した英文字であっても、記憶回路には大文字に変換されて記憶されますから、あえて大文字で入力する必要はありませんが、大文字で入力しなければならない場合があります。

大文字で入力する場合は、[CAPS]（キャピタルロック）と書かれたキーを押してロックします。

CAPS
↑
キーをロックして入力すると大文字で入力される

すると、叩いたキーの英文字は、上に示すように大文字で画面に表示されます。

[CAPS]キーをロックして、大文字で入力したあと、ふたたび小文字の入力に戻すには、もう一度、[CAPS]キーを押します。すると[CAPS]キ



大文字の入力

ーのロックが解除されて、小文字の入力に戻ります。また、CAPSキーをロックして大文字で入力しているときに、SHIFTキーを押しながら英文字のキーを叩くと、小文字で入力することができます。

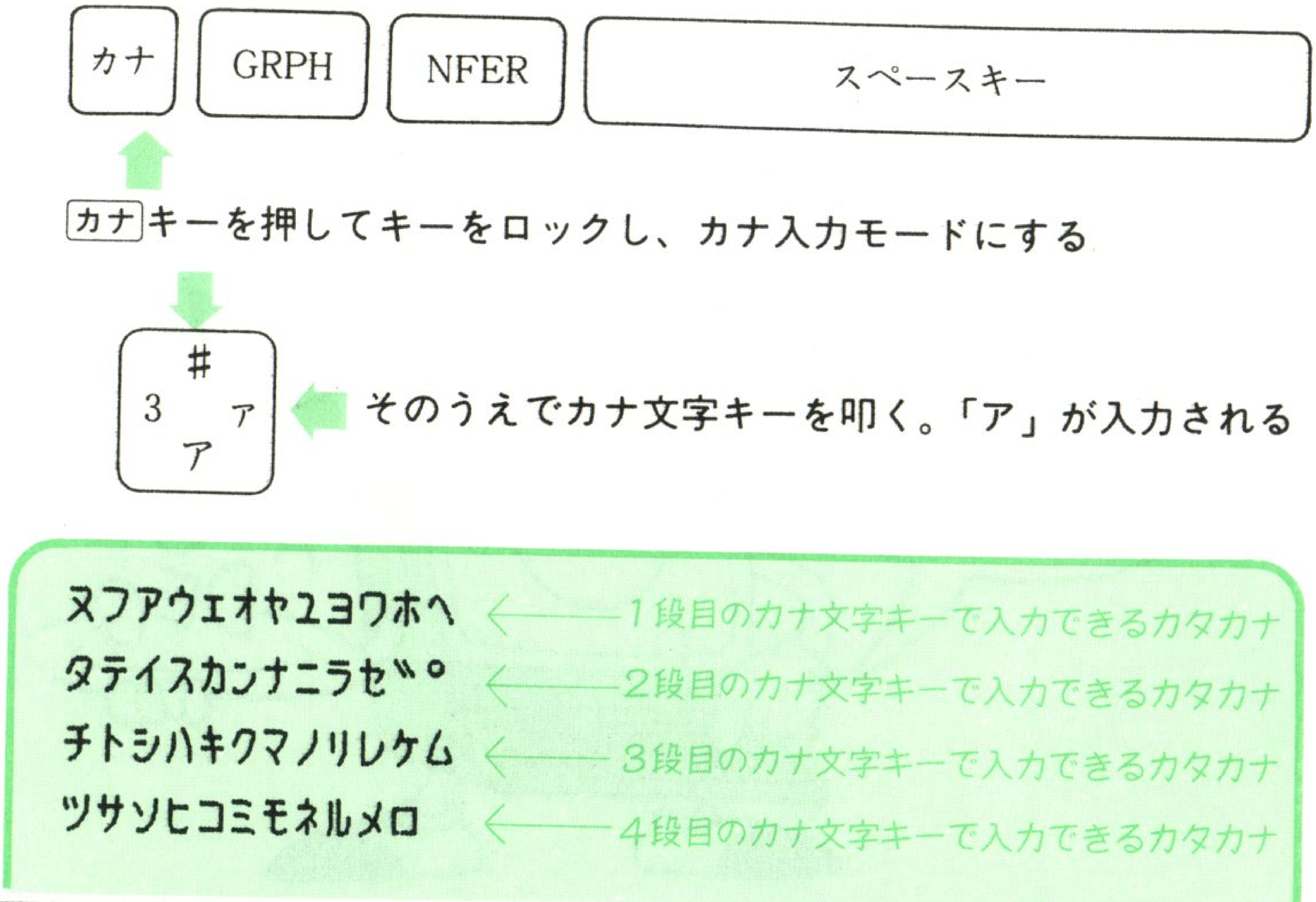
CAPSキーは、下から3段目の列の左から2番目にあります。

なお、CAPSキーをロックせずに、英文字を大文字で入力することができます。その場合は、SHIFTキーを押しながらアルファベットのキーを叩くと、英文字が大文字で入力されます。

カナ文字キーはカタカナの入力

もっとも頻繁に使う英文字と数字、それに記号の入力の仕方がわかったところで、つぎはカタカナの入力です。

カタカナを入力するには、アからンまでのカタカナが示されている45個のカナ文字キーと、それに「〃(濁点)」、「。(半濁点)」、「ー(長音)」などの



カタカナの入力

キーを叩いて行います。

ところで、カナ文字キーはアルファベットのキーや数字キー、記号キーなどとおなじキーにありますから、そのままキーを叩くと英文字や数字、記号などが入力されてしまいます。

そのため、カタカナを入力する場合は、キーボードの一番下の段、左端にある「カナ」と書かれたキーを押して[カナ]キーをロックし、カナ入力モードにします。



そのうえでカナ文字キーを叩くと、叩かれたキーのカタカナが入力されます。

「ジ」、「プ」などの入力

パソコンでは、ワープロとは異なって、単にカナ文字キーを叩いた場合、「ジ」や「バ」というような「〃」濁点、「プ」や「ポ」というような「。」半濁点をともなったカタカナを1文字で入力することができません。

濁点をともなったカタカナを入力するには、つ

［２ フ］ ← カナ文字キーを叩く

［＠ ゛］ ← 濁点キーを叩く

ﾊﾞﾋﾞﾌﾞﾍﾞﾎﾞｶﾞｷﾞｸﾞｹﾞｺﾞﾀﾞｼﾞｽﾞ

［－ ホ］ ← カナ文字キーを叩く

［［ 「 。］ ← 半濁点キーを叩く

ﾊﾟﾋﾟﾌﾟﾍﾟﾎﾟ

濁点と半濁点をともなったカタカナの入力

ぎのように、カナ文字キーを叩いてから「〃」濁点キー（Ｐキーの右のキー）を叩きます。

［シ］ ← ［〃］ ← ［ｼ］［ﾞ］

↑カナ文字キーを叩く　↑濁点キーを叩く

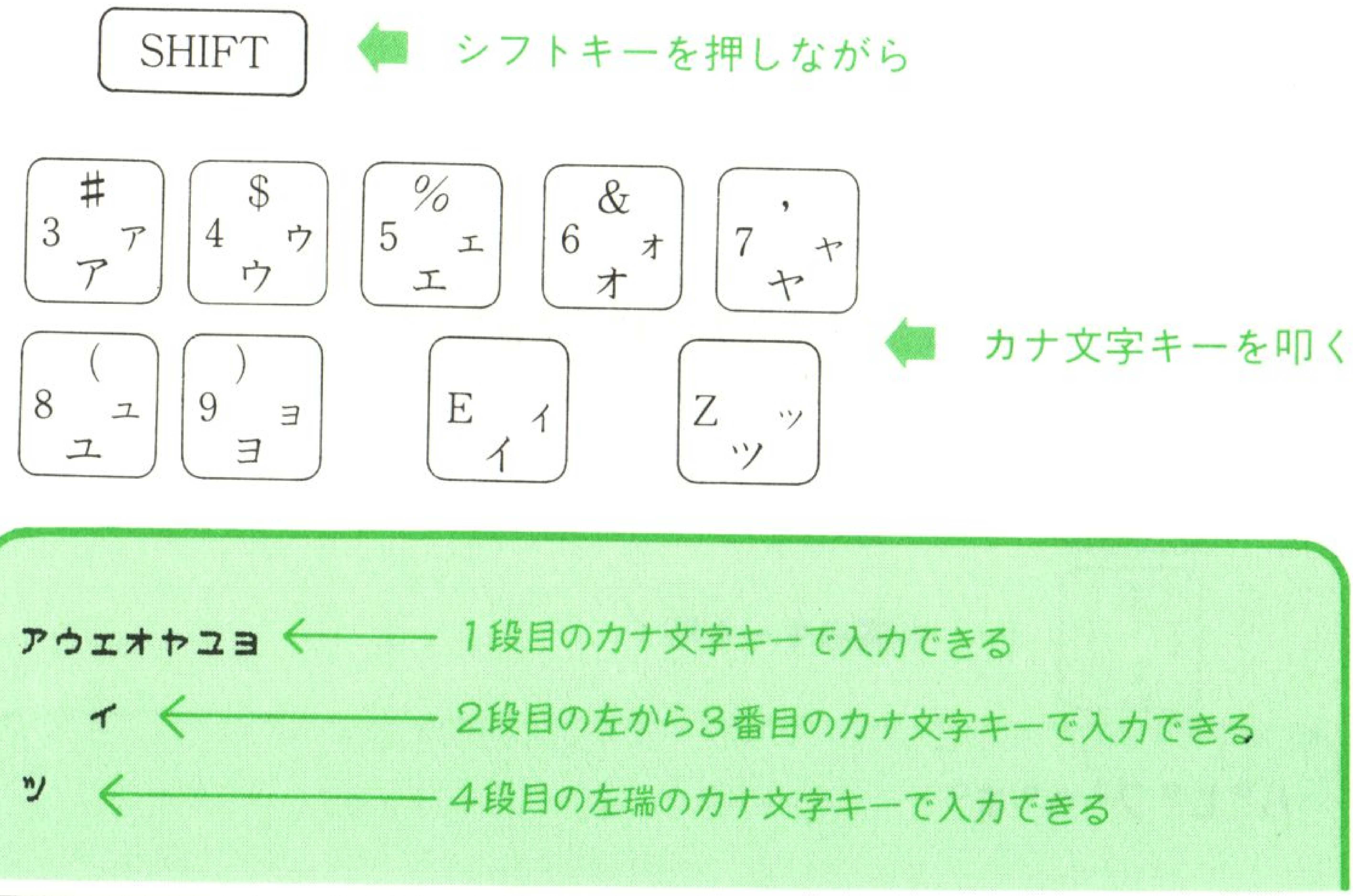
このように、濁点をともなったカタカナは、カナ文字キーと濁点キーの二つのキーで入力されますから、数えるときは、二文字として数えることになります。

半濁点をともなったカタカナの場合も、カナ文字キーを叩いてから「。」半濁点キー（Ｐキーの二つ右のキー）を叩きます。

［ハ］ ← ［。］ ← ［ﾊ］［ﾟ］

↑カナ文字キーを叩く　↑半濁点キーを叩く

半濁点をともなったカタカナも、濁点をともな



促音の入力

ったカタカナとおなじように、二文字として数えます。

なお、半濁点キーは一般キーの2段目、右端にあります。濁点キーは、半濁点キーの左にあります。それぞれのキーの形は、まえの頁の図に示すような形です。

■促音を入力するには

カタカナの入力の場合、促音を入力することがあります。

促音とは、たとえば「イッタ」というような場合の「ッ」のことをいいます。

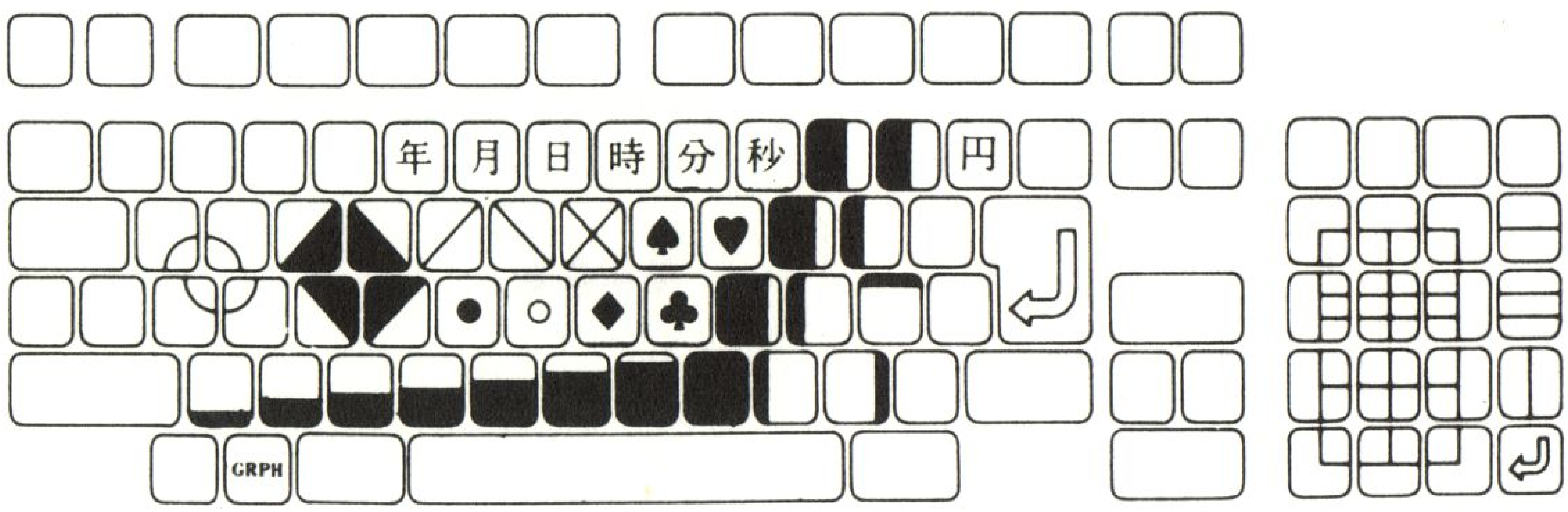
促音は、つぎのように九つあります。

ァゥェォャュョィッ←促音

九つの促音を示す小さいカタカナは、うえに示すように、それぞれ「ァゥェォャュョィッ」のカナ文字キーの右側に示されていますから、ただキーを叩くだけでは入力することができません。

[SHIFT]キーを押しながら、「ァゥェォャュョィッ」のいずれかのキーを叩くと、入力されるようになっています。





グラフィックシンボルの配置

GRPH(グラフ)キーは グラフィックシンボルの入力

どのパソコンでも、英文字、数字、記号、カタカナのほかに、グラフィックシンボルを入力することができます。

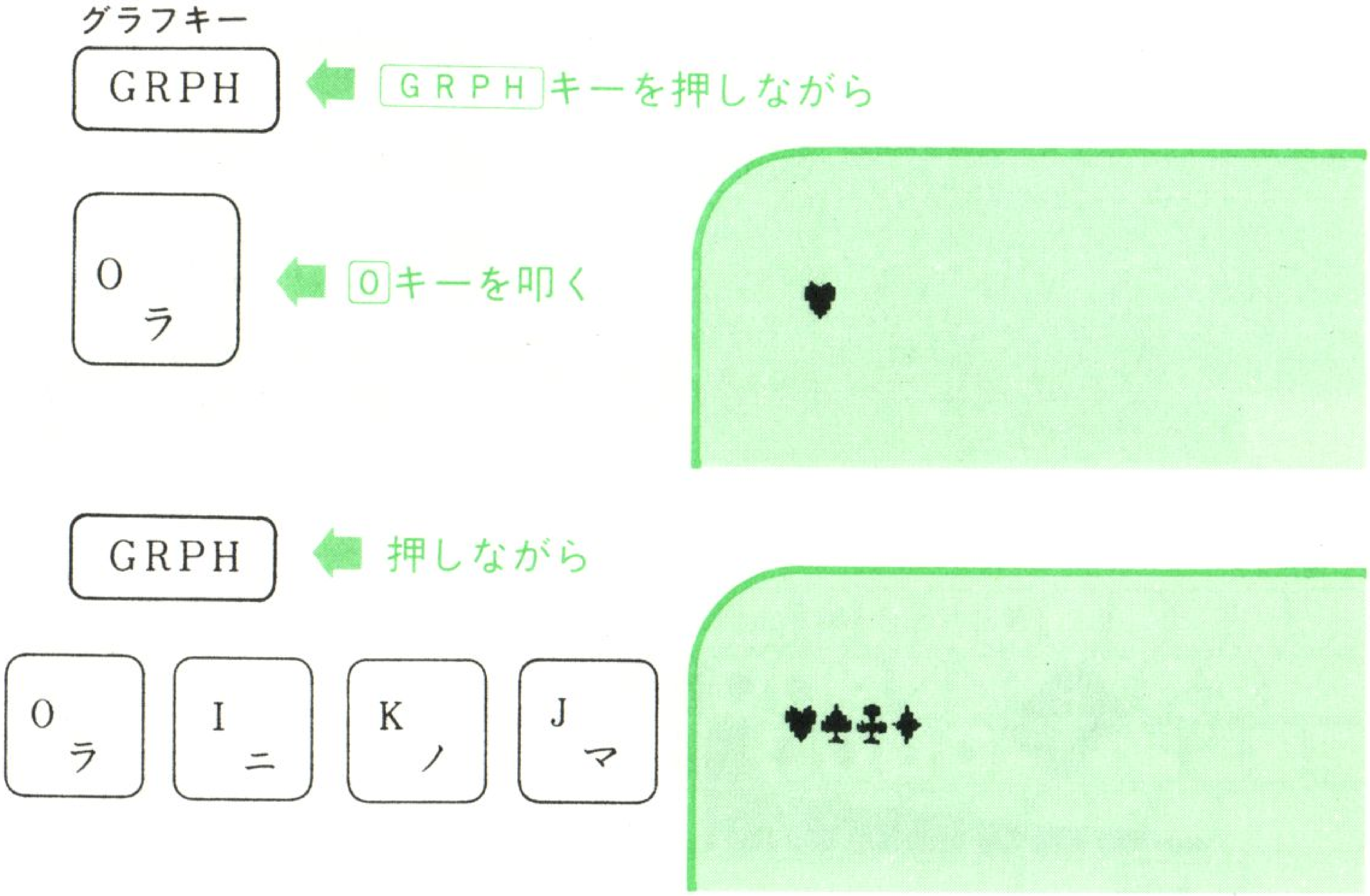
グラフィックシンボルは、アルファベットやカタカナなどのように、キーの表面には示されていませんが、上に示すように、グラフィックシンボルが配列されています。

このグラフィックシンボルを入力する場合は、カナキーの右隣りにあるGRPHと書かれたグラフィックキーを押しながら、入力するグラフィックシンボルがある位置のキーを押します。

GRPH
↑
グラフィックシンボルを入力するときに押す

GRPHキーは、カナキーのようにロックすることができませんから、注意してください。

たとえば、グラフィックシンボルの「♥」ハートを入力するとすれば、「♥」は一般キーの二段目



グラフィックシンボルの入力

の、RETURNキーから数えて五つ目にあります。五つ目のキーは、アルファベットのOキーです。

したがって、GRPHキーを押しながら、このOキーを叩くと、

GRPH ↑押しながら
＋
O ↑叩く
♥が入力される

上に示すように「♥」が入力されます。

いろいろなグラフィックシンボルを同時に入力する場合は、GRPHキーを押しながら、それぞれのキーを叩けばよいのです。

たとえば、「♥」、「♠」、「♣」、「♦」を入力するときは、GRPHキーを押しながら、アルファベットのOキー、Iキー、Kキー、Jキーをそれぞれ叩くと、上に示すように、入力することができます。

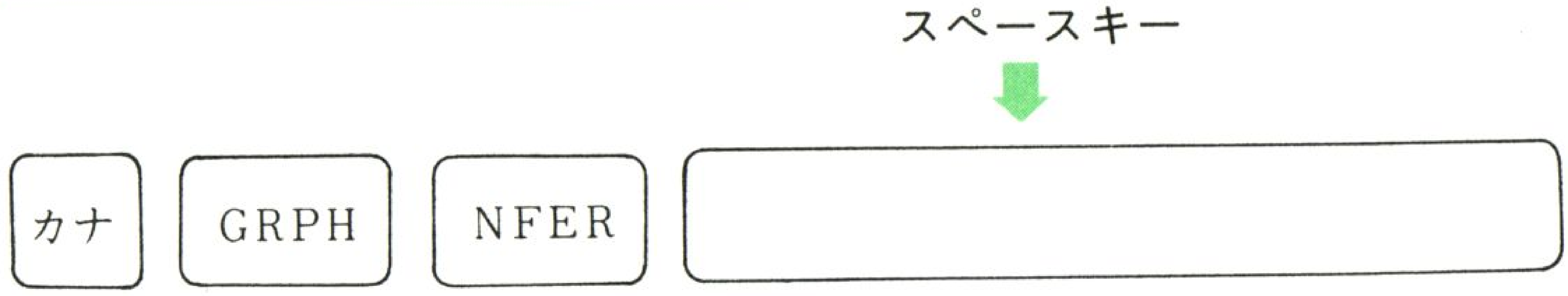
GRPH ↑押しながら
＋
O I K J ↑叩く
↓
♥♠♣♦

```
10 PRINT 10+10
```

スペースキーを一回叩いて空白を入力

```
10 PRINT "コタエ ハ ";A;" デース"
```

スペースキーを叩いて空白を入力



スペースキーは空白を入力

スペースキーは空白を入力

キーボードの一番下の段、つまり[カナ]キーや、[GRPH]キーがある列に、なにも文字が書かれていない細くて長いキーがあります。このキーを、スペースキーといいます。

スペースキーは空白を入力するキーで、スペースキーを一回叩くと空白が一つ、二回叩くと空白が二つ入力されます。

空白は、いろいろな場合に入力します。たとえば、上に示すように、行番号とベーシックの命令の間に入力したり、ベーシックの命令と計算式の間に入力したり、また、画面に表示する文字が見やすいように、文字と文字の間に入力したりします。

このように入力した空白は、文字や数字、記号などとおなじように扱われます。したがって、数える場合は、スペースも一文字として数えることになっています。

```
10 CLS
20 LOCATE 15,10
30 PRINT "ニュウリョク シタ デ゛ータ ノ コ゛ウケイ ヲ モトメ マス"
40 LOCATE 15,12
50 PRINT "オワリ ノ トキハ 0 ヲ ニュウリョク シテクタ゛サイ"
60 FOR T=0 TO 9000 :NEXT:CLS
70 INPUT "デ゛ータ ヲ イレテ クタ゛サイ";A
80 IF A=0 THEN END
90 B=B+A
100 PRINT "コ゛ウケイ = ";B
110 GOTO 70
```

データの合計を求めるプログラム

ベーシックのプログラムを入力してみよう

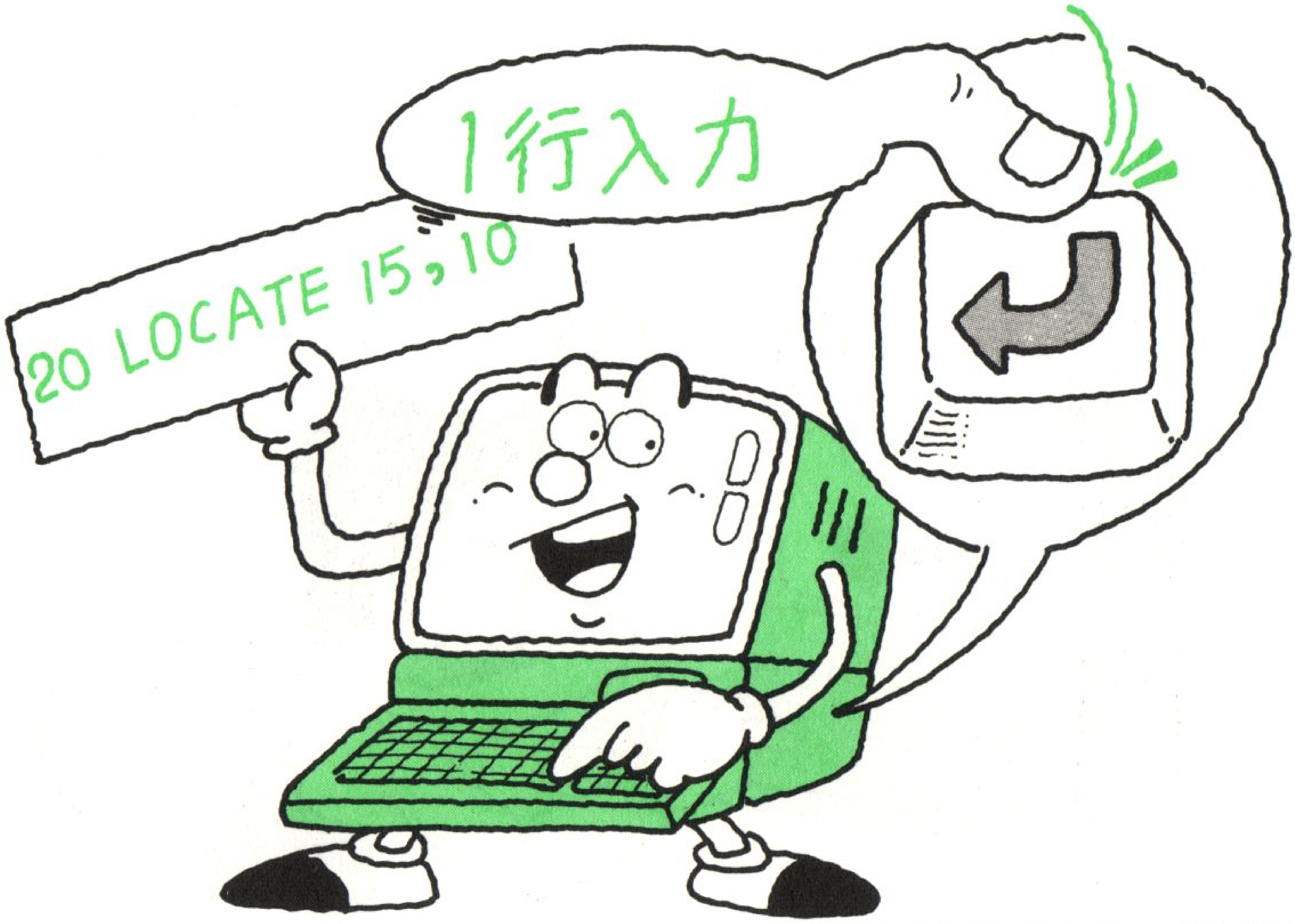
これまで画面をクリアするキーから、英文字、数字、記号、カタカナ、空白を入力するキーについて示しました。これらのキーについてわかれば、もうプログラムやデータの入力ができます。

そこでキー操作の練習のために、プログラムを入力してみることにします。ここで入力するプログラムを、上に示します。

ここではキー操作の練習ですから、プログラムの一つ一つの命令については省略します。

プログラムの入力の仕方

プログラムを入力するには、キーボードのキーを叩いてベーシックの命令を一行入力し終わった



ベーシックの命令を１行入力したら⏎キーを叩くこと

ら、⏎キーを叩きます。

10 CLS⏎　←一行入力し終わったら⏎キーを叩く

するとカーソルは、つぎの行の先頭に移動して点滅しますから、つぎのベーシックの命令を入力します。入力し終わったら⏎キーを叩きます。

20 LOCATE 15, 10⏎　←⏎キーを叩く

このことを繰り返して、プログラムを入力していきます。

ところで、ベーシックの命令を一行入力するたびに⏎キーを叩くのは、⏎キーを叩くことによって、ＣＰＵに、ここまでがベーシックの命令の一行分であることを知らせるとともに、入力した一行分の命令を記憶場所に記憶させるよう指示しているわけです。

つまり39頁で示したように、ＣＰＵはプログラムを記憶場所に記憶したうえで、その命令を記憶場所から一つ一つ取り出して実行していきますから、キーボードのキーを叩いて入力したベーシックの命令が画面に表示されたからといって、ベーシックの命令を入力したことにはなりません。



キーの叩き間違いなど気にしないでキーを叩く

キーボードのキーを叩いて入力したベーシックの命令を、⏎キーを叩いて記憶場所に記憶させてはじめて、本当の意味でベーシックの命令を入力したということになります。

ベーシックの命令は間違えて入力してもかまわない

ところで、プログラムを入力していくときに、「命令を間違えて入力しては」などと気を使ってキーを叩いていたのでは、キー操作が思うようにいきません。キーを叩き間違えて、プログラムの命令が間違って入力されてもパソコンは故障したりしませんから、そんなことに頓着しないで、どんどんキーを叩いて入力していくことです。

どんなにパソコンを扱い慣れている人であっても、キーを1文字も叩き間違えることなく、プログラムの命令を入力するなどということは、まずありません。

パソコンを扱い慣れている人とは、プログラムを間違えて入力することなど当たりまえと思って



ベーシックは対話型の言語

いる人といえなくもありません。

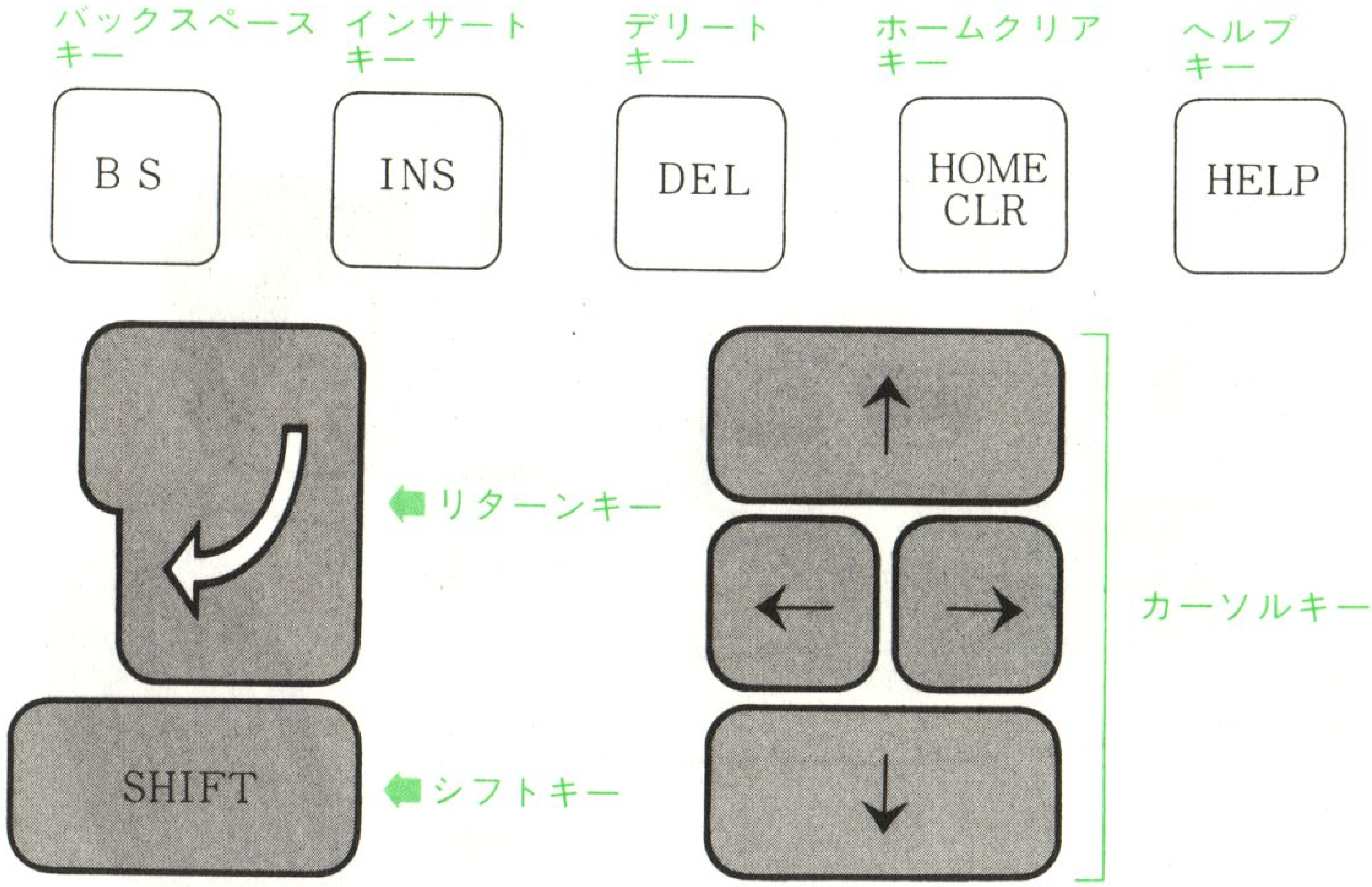
ベーシックは対話型だから

プログラムの入力に間違いがあってもかまわずに、キー操作に慣れるためにどんどん入力してみると言ったのは、ベーシックは、対話型というプログラミング言語だからです。

対話型のプログラミング言語というのは、私たちがパソコンと対話しながらプログラムを作っていくという形式の言語です。

したがって、プログラムが間違っていようとなかろうと、ともかくプログラムを入力してしまって、実行させます。

もし、入力したプログラムに文法上の誤りや命令が間違っているところがあるなら、パソコンのほうでその間違いを見つけて、エラーメッセージを表示して、どこそこが間違っていると私たちに教えてくれますから、間違っているところを訂正すればよいのです。



入力したプログラムを訂正するために使うキー

入力したプログラムの間違いを訂正する

では、入力したプログラムの間違いを訂正するためのキーボードの操作です。

入力したプログラムの間違いを訂正するために使うキーは、上に示すように[HELP]（ヘルプ）キー、カーソルキー、[DEL]（デリート）キー、[BS]（バックスペース）キー、[INS]（インサート）キー、[⏎]（リターン）キーです。

では82頁に示したプログラムを実行させて、プログラムの訂正を実際に行いながら、これらのキーの使い方について示すことにします。

まず、入力したプログラムを実行させます。

プログラムを実行させるコマンドは、RUNコマンド（命令）です。

RUN ＝ プログラムを実行させる命令

```
ニュウリョク シタ デ゛ータ ノ ゴ゛ウケイ ヲ モトメ マス

オワリ ノ トキハ 0 ヲ ニュウリョク シテクダ゛サイ
```

```
デ゛ータ ヲ イレテ クダ゛サイ?■
```

入力したプログラムの実行

ＲＵＮコマンドを入力するには、キーボードの一番上の段の中央にある[f・5]と書かれたキー、ファンクションキーを押します。

[f・5] ← RUNコマンドを入力

このキーを押すだけでＲＵＮコマンドが入力されて、プログラムが即座に実行されます。

ＲＵＮコマンドを入力するとプログラムが実行されて、上に示すように、このプログラムはデータの合計を算出するプログラムであるというガイド文が表示されます。

このガイド文が消えると、つぎにデータの入力を求めるガイド文と？マークが表示されますから、つぎのようにデータを一つ一つ入力しては、そのたびに[⏎]キーを叩くと、入力したデータの合計が算出されて表示されます。

データ　ヲ　イレテ　クダサイ？　100 [⏎]
データ　ヲ　イレテ　クダサイ？　200 [⏎]
データ　ヲ　イレテ　クダサイ？　300 [⏎]

ところで、入力したプログラムに間違いがあると、つぎの頁に示すように、エラーメッセージを

```
ﾆｭｳﾘｮｸ ｼﾀ ﾃﾞｰﾀ ﾉ ｺﾞｳｹｲ ｦ ﾓﾄﾒ ﾏｽ
Syntax error in 40
Ok
■
```

Syntax error in 40 ← ベーシックの命令に間違いがあるとエラーメッセージを表示

```
ﾆｭｳﾘｮｸ ｼﾀ ﾃﾞｰﾀ ﾉ ｺﾞｳｹｲ ｦ ﾓﾄﾒ ﾏｽ
Syntax error in 40
Ok

40 LOCCATE 15,12
```

カーソル（「15」の「1」の位置）

40 LOCCATE 15,12 ← HELPキーを押すとエラーメッセージで指摘された命令だけを画面に表示する

エラーメッセージとHELPキーの働き

表示して、プログラムの実行を中止します。

エラーメッセージにはいろいろありますが、

Syntax error in 行番号

この**Syntax error**（シンタックスエラー）が、一番多い間違いです。**Syntax error in 40**とは、行番号40の命令に文法的な間違いがあるということです。

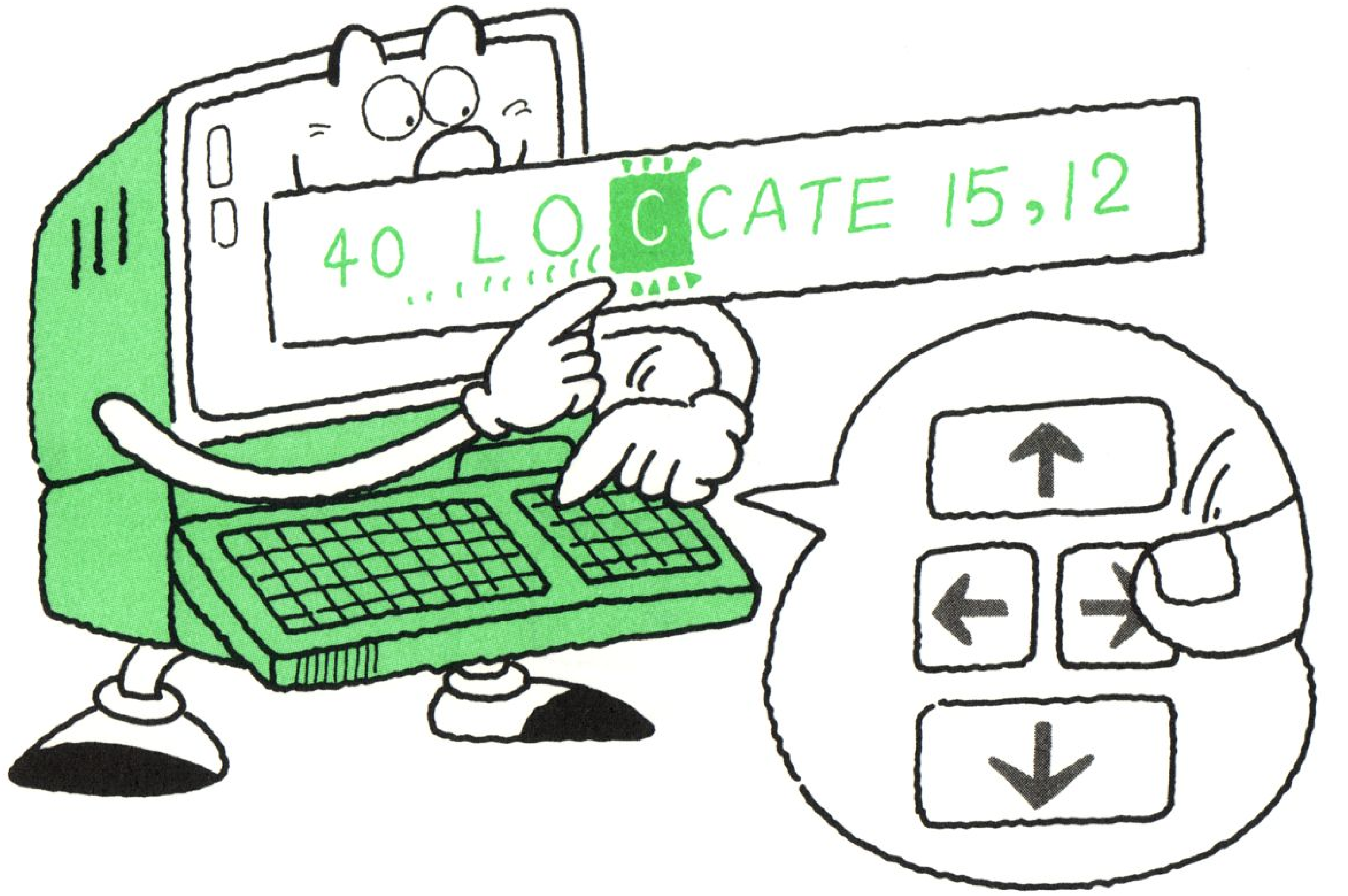
HELPキーは間違いがある行番号の命令を画面に表示

では、Syntax error in 40というエラーメッセージで指摘された行番号40の命令にある間違いの訂正です。

間違いを訂正するには、まず行番号40の命令を画面に表示させます。この場合に使うのが、HELP（ヘルプ）キーです。

HELP ＝ 間違いのある命令を画面に表示

HELPキーを押すと、エラーメッセージで間違いがあると指摘された行番号40の命令だけが、上に示すように画面に表示されます。



カーソルキーでカーソルを訂正する箇所まで移動

カーソルキーはカーソルを移動

行番号40の命令が画面に表示されたら、間違っている箇所を捜します。行番号40の正しい命令の形は、つぎのようになります（ＬＯＣＡＴＥ（ロケート）は159頁参照）。

40 LOCATE 15, 12

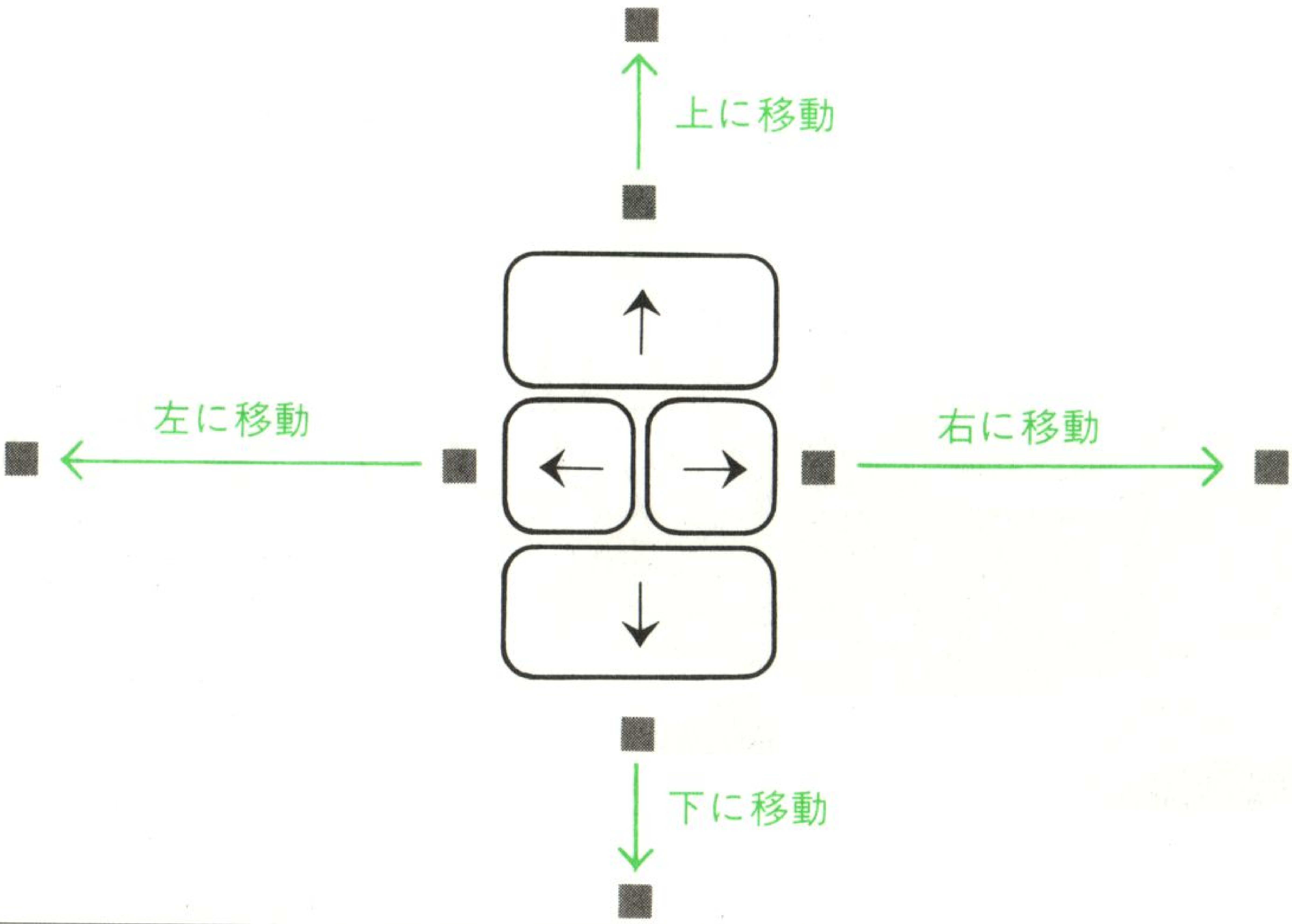
ところが、画面に表示された命令をみると、

40 LOCCATE 15, 12

Cが一つ余分にキー入力されています。

この余分なCを削除しなければなりませんが、単に削除するキーを押しても、Cを削除することはできません。というのは、削除や挿入ばかりでなく、間違いを訂正するには、カーソルをまず訂正するところまで移動させなければならないからです。そこで、Cを削除するには、カーソルを削除するCの位置まで移動させてから、削除するキーを押します。

さて、カーソルを移動させるには、カーソルキ



カーソルキーとカーソルが移動する方向

ーを使います。

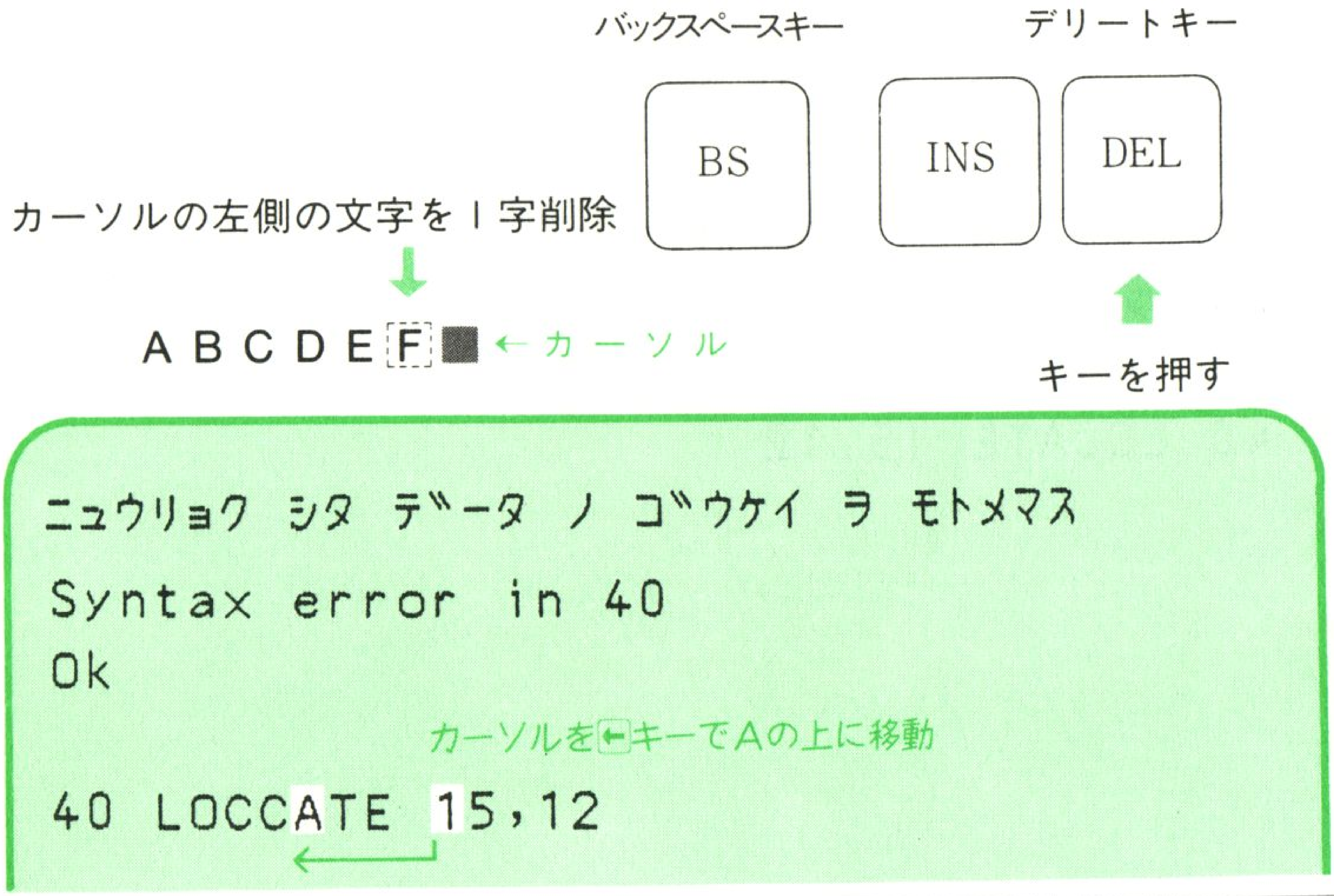
カーソルキー＝カーソルを移動させる

カーソルキーは4つあり、それぞれのキーには矢印が書かれています。[→]、[↓]、[←]、[↑]の矢印は、カーソルが移動する方向を示しています。

カーソルキーを一回押すと、押した矢印の方向に向けて、カーソルは一文字分移動します。カーソルキーを押しつづけると、矢印の方向に向けてカーソルは連続して移動していきます。

さて、[HELP]キーを押して間違いのある命令が画面に表示されると、88頁で示したように、カーソルは行番号40の命令の15の1のうえに表示されます。そして削除するCは、カーソルが表示されている位置より左にあります。

したがって、一つ余計なCを削除するには、[←]キーを使ってカーソルを左に移動させることになりますが、カーソルを左に向けてどこまで移動させるかは、削除するキーとの関係があるので、つぎの[DEL]キー、[BS]キーのところで示します。



文字の削除(1)

DEL、BSキーは文字の削除

文字を削除するキーは、DEL（デリート）キーです。BS（バックスペース）キーも文字を削除するキーですが、ふつうDELキーを使います、

DEL　BS　← 一文字を削除

DELキーを一回押すと、カーソルの左側にある一文字が削除されます。BSキーを一回押した場合も、カーソルの左側にある一文字が削除されます。

このように、文字を削除するDELキーは、カーソルの左側にある一文字を削除しますから、さきの余計なCを削除するには、上に示すように、←キーでカーソルをAの文字の上か、あるいはCの文字の上に移動させます。

なぜなら、二つのCのうち、どちらのCを削除しようとかまわないからです。要は、命令のなかのCを一つにすればよいのです。

```
ﾆｭｳﾘｮｸ ｼﾀ ﾃﾞｰﾀ ﾉ ｺﾞｳｹｲ ｦ ﾓﾄﾒﾏｽ
Syntax error in 40
Ok

40 LOCATE 15,12
```

DELキーを押すと、カーソルが表示されているAの左側のCが削除される。

文字の削除(2)

[←]キーで、Aの文字の上にカーソルを移動させて[DEL]キーを押した場合は、上に示すように、Aの文字の左側のCが削除されます。また、Cの文字の上にカーソルを移動させて、[DEL]キーを押した場合は、Cの左側のCが削除されます。

なお、続けて文字を削除するときは、削除する文字数に応じて、[DEL]キーを繰り返し押せばよいのです。

訂正が終わったら[⏎]キーを叩く

[DEL]キーで余計なCの文字を削除したら、そのままの状態で[⏎]キーを叩き、Cの文字削除して訂正したベーシックの命令を新たに入力します。

このように[⏎]キーを叩いて、訂正したベーシックの命令を新たに入力しないと、Cの文字を削除するまえのベーシックの命令が、そのまま記憶場所に記憶されて残っていますからふたたびSyntax error in 40 が表示されてしまい、Cの文字を削除して命令を訂正したことにならないので注意し

```
ニュウリョク シタ データ ノ ゴウケイ ヲ モトメ マス
Syntax error in 40
Ok

40 LOCATE 15,12 ⏎
40 LOCATE 15,12
```

40 LOCATE 15,12 ⏎ ←—— 削除したら⏎キーを叩いて改めて入力する

40 LOCATE 15,12 ←—— HELPキーを押すと訂正した行番号40を表示するので確認する

削除が終わったら⏎キーを叩いて命令を入力

てください。

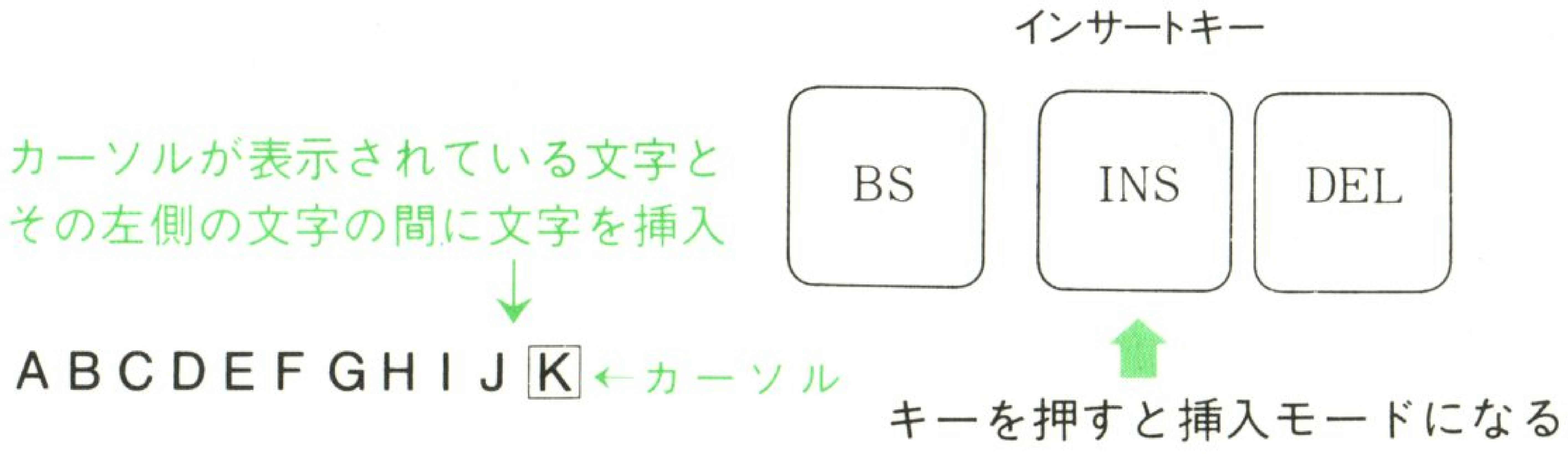
もちろん文字の削除ばかりではなく、どんな訂正の場合でも、訂正が終わったら必ず⏎キーを叩いて、訂正したベーシックの命令を新たに入力しなければなりません。

INSキーは文字を挿入

さて⏎キーを叩いて、Cを削除して訂正したベーシックの命令を新たに入力したら、ふたたびRUNコマンドを入力してプログラムを実行させます。

間違いのあった行番号40のベーシックの命令を訂正したのでエラーメッセージは出ず、87頁に示したように2行のガイド文が消えると、データの入力を求めるガイド文と?マークが表示されます。

ここでは、データとして1000を入力することにしますから、1000とテンキーを叩いてから⏎キーを叩いて入力します。




```
デ゛ータ ヲ イレテクダ゛サイ? 1000 ⏎
Syntax error in 100
Ok

100 PINT "コ゛ウケイ = ";B
```

←1000を入力

←エラーメッセージを表示

カーソル（"の位置）

←HELPキーを押すと間違いのある命令を表示

文字の挿入(1)

さて、⏎キーを叩いて1000を入力すると、上に示すようにふたたびエラーメッセージが表示されました。そこで、HELPキーを押して、行番号100のベーシックの命令を画面に表示させます。

行番号100の正しい命令は、つぎのようになります。

100 PRINT "ゴウケイ = ";B

HELPキーを押して表示させたベーシックの命令は、つぎのように、

100 PINT "ゴウケイ = ";B

PRINTのRが抜けていますから、抜けているRを挿入しなければなりません。

文字を挿入するキーは、INS（インサート）キーです。

INS = 文字などを挿入

INSキーを一度押すと挿入モードになって、カーソルが表示されている位置にある文字と、その左側にある文字との間に、文字を挿入することができます。

一度挿入モードにすると、挿入モードを解除し

```
デ゛ータ ヲ イレテクダ゛サイ? 1000
Syntax error in 100
Ok

100 PINT "コ゛ウケイ = ";B
```

←キーでカーソルをIの上に移動

```
デ゛ータ ヲ イレテクダ゛サイ? 1000
Syntax error in 100
Ok

100 PrINT "コ゛ウケイ = ";B
```

カーソル

INSキーを押してRのキーを叩く

文字の挿入(2)

ないかぎり挿入モードがつづくので、何文字でも挿入することができます。

挿入モードを解除するには、もう一度[INS]キーを押すか、[←]あるいは[→]キーでカーソルを左か右に移動させればよいのです。

ところでこの場合は、Pの文字とIの文字の間にRを挿入するわけですから、[←]キーでカーソルを左に移動させて、上に示すように、Iの文字の上にカーソルをもってきます。

カーソルをIの文字の上にもってきたら、[INS]キーを押します。これで挿入モードになりましたから、挿入する文字である[R]キーを叩きます。

すると、上に示すようにRの文字が、カーソルが表示されている位置にあるIの文字と、その左側にあったPの文字との間に挿入されます。

このようにRの文字を挿入したら、[↵]キーを叩いて訂正した命令を新たに入力しますが、この場合、挿入モードを解除してから[↵]キーを叩いても、また、訂正した状態のままで[↵]キーを叩いても、どちらでもかまいません。

```
ﾃﾞｰﾀ ｦ ｲﾚﾃ ｸﾀﾞｻｲ? 100
Syntax error in 100
Ok

100 PRENT "ｺﾞｳｹｲ = ";B
```

←キーでカーソルを間違っている文字の上に移動

```
ﾃﾞｰﾀ ｦ ｲﾚﾃ ｸﾀﾞｻｲ? 100
Syntax error in 100
Ok

100 PRiNT "ｺﾞｳｹｲ = ";B⏎
```

訂正したら⏎キーを叩いて入力する

Iキーを叩いてEをIに訂正する

間違っている文字の訂正

間違っている文字の訂正

これまで、文字の削除と文字の挿入について示しましたが、入力の間違いのなかでもっとも多いのは、キーボードのキーを叩き間違えて、別の文字を入力してしまうことです。

たとえば、画面に表示させる命令は、

PRINT

が正しいわけですが、上に示すように、キーボードのキーを叩き間違えて、

PRENT

IをEと入力してしまうなどです。

このような文字の間違いを訂正するには、まず←キーでカーソルを左に移動させて、訂正する文字Eの上に持ってきます。

カーソルを訂正する文字Eの上に持ってきたらIキーを叩きます。すると、EがIに変更されますから、つぎに⏎キーを叩いて、訂正した命令を新たに入力します。

f・1	f・2	f・3	f・4	f・5
load"	auto	go to	list	run^{CR}

f・6	f・7	f・8	f・9	f・10
save"	key	print	edit .	cont^{CR}

ファンクションキーと内容の表示

ファンクションキーで命令やコマンドを入力

画面の下に白い四角い箱が表示されていて、そのなかに、load",autoなどの文字が表示されていますが、これはファンクションキーの内容を示しています。ファンクションキーとは、キーボードの一番上の段にある［f・1］キーから［f・10］までのキーです。

［f・8］キーはPRINTを入力

ファンクションキーの内容の表示とファンクションキーの［f・1］から［f・10］までは、上に示すように対応していて、［f・1］から［f・10］までのキーを押すと、内容として表示されているベーシックの命令やコマンドが入力されます。

```
10 print ← f・8キーを押して入力
```

```
     f・5キーを押すとプログラムが実行される
run ↙
ニュウリョク シタ デ゛ータ ノ ゴ゛ウケイ ヲ モトメ マス
オワリ ノ トキハ 0 ヲ ニュウリョク シテクダ゛サイ
```

ファンクションキーの働き(1)

したがって、f・8キーは内容の表示にPRINTとありますから、f・8キーを押すと、上に示すように、ベーシックの命令のPRINT（画面に表示せよ）が入力されることになります。

つまり、PRINTを入力するときは、PRINTというようにキーを5回叩かなくても、f・8キーを1回押すだけで、入力することができるというわけです。

f・5キーはプログラムを実行

f・5キーは、内容の表示ではrunCRとなっています。run（ラン）は、プログラムを実行させるコマンド（命令）です。

run＝プログラムを実行させるコマンド

このrunには、PRINTの場合と違ってCRがついていますが、このCRは、⏎キーも同時に入力されるという意味です。

つまり、キーボードのキーを叩いてrunと入力したときは、つぎのように⏎キーを叩かなけれ

```
list ⏎  ←―――― f・4キーを押して⏎キーを叩く
10 CLS
20 LOCATE 30,10:PRINT "ｴﾝﾊﾞﾝ ｦ ﾄﾊﾞｼﾏｽ"
30 FOR T=0 TO 5000:NEXT:CLS
40 Y=INT(RND(1)*15)+1
50 FOR X=0 TO 70
60 LOCATE X,Y:PRINT "▲"
70 FOR T=0 TO 80:NEXT
80 LOCATE X,Y :PRINT "  "
90 NEXT
100 GOTO 40
Ok
```

メモリに記憶されたプログラムを画面に表示

ファンクションキーの働き(2)

ばなりませんが、

run⏎ ← runと入力して⏎キーを叩く

f・5キーはただ押すだけで、まえの頁に示すようにプログラムが実行されます。

f・4キーはメモリの情報を画面に表示

f・4キーは、内容の表示ではlistとなっています。LIST(リスト)とは、メモリ(記憶回路)に記憶されているプログラムを画面に表示させるコマンドです。

LIST＝メモリのなかの情報を画面に表示

このLISTには、RUNのようにCRがついていませんから、f・4キーを押したら⏎キーを叩きます。

LIST⏎ ← f・4キーを押して⏎キーを叩く

すると上に示すように、メモリに記憶されているプログラムなどが画面に表示されます。

上に示すプログラムは、エンバンを飛ばすプログラムです。

ストップキー



←ストップキーを押すとプログラムの実行を停止

```
                                ▲
^C          ←―――― STOPキーを押したとき入力される記号
Break in 70 ←―――― 行番号70の命令で実行を停止したことを示す
Ok          ←―――― ほかのコマンドの入力を受けつける
■
```

プログラムの実行を停止（STOPキー）

プログラムの実行と画面表示の停止

プログラムを実行させるには、RUNコマンドを使いますが、プログラムの実行中に、プログラムの実行を停止したり、画面表示を一時停止したり、一時停止したプログラムを再び実行させることがよくあります。

STOPキーは実行を停止

プログラムの実行を停止させるには、STOPキーを使います。

STOP ＝ 実行を停止

たとえば、まえの頁に示したプログラムを実行させると、▲が画面の左から右に向けて移動していきます。STOPキーを押すと、上に示すよう

```
list⏎  ←―――― メモリのプログラムを画面に表示させる
10 CLS
20 LOCATE 30,10:PRINT "ｴﾝﾊﾞﾝ ｦ ﾄﾊﾞｼﾏｽ"
30 FOR T=0 TO 5000:NEXT:CLS
40 Y=INT(RND(1)*15)+1
50 FOR X=0 TO 70
^C  ←―――― STOPキーを押すと入力される
Break ←―――― 表示の停止を示す
Ok ←―――― コマンドレベル
■
```

画面表示の停止（STOPキー）

にメッセージなどを表示して、プログラムの実行を中止します。

実行を中止するとコマンドレベルに戻る

プログラムの実行を中止すると、最後に必ずOKが表示されます。このOKは、コマンドの入力を受けつけるという印ですから、ふたたびRUNコマンドを入力して、プログラムを実行させたり、また、LISTコマンドを入力してメモリのプログラムを画面に表示させたり、LLISTコマンドで、プログラムをプリンタから印字して打ち出すこともできます。

このようにOKが表示されている状態は、コマンドの入力を受けつける状態であることから、これを**コマンドレベル**といいます。

もちろんSTOPキーは、プログラムの実行を中止させるばかりではありません。LISTコマンドで、メモリに記憶されているプログラムを画面に表示させた場合でも、STOPキーを押すと、上に示すようにメッセージを表示して、画面への表示を停止します。

コントロールキー

CTRL ＋ S

```
list⏎
10 CLS
20 LOCATE 30,10:PRINT "ｴﾝﾊﾞﾝ ｦ ﾄﾊﾞｼﾏｽ"
30 FOR T=0 TO 5000:NEXT:CLS
40 Y=INT(RND(1)*15)+1
50 FOR X=0 TO 70
```

← LISTコマンドでプログラムを画面に表示

CTRLキーを押しながらSキーを押したところで画面への表示を一時停止する

← 一時停止のときはカーソルも表示されない

CTRL＋S キーは実行を一時停止する

CTRL＋Sキーは実行を一時停止

LISTコマンドで、メモリのプログラムを画面に表示させる場合、プログラムが長くて画面に収まりきらないと、最初に画面に表示されたほうから順に画面の上のほうから消えていきます。

このように画面から順に消えていくことを、**スクロール**といいます。

ところでスクロールされてしまっては、プログラムの最初のほうを見ることができませんから、画面に一杯になるまえに画面への表示を一時停止させます。

一時停止させるには、CTRLキーを押しながら英文字のSキーを叩きます。

CTRL＋S＝画面表示を一時停止

すると、上に示すように、画面への表示が一時停止されます。

一時停止の場合は、STOPキーを押した場合と違って、メッセージやカーソルも表示されませ

```
list↵
10 CLS
20 LOCATE 30,10:PRINT "ｴﾝﾊﾞﾝ ｦ ﾄﾊﾞｼﾏｽ"
30 FOR T=0 TO 5000:NEXT:CLS
40 Y=INT(RND(1)*15)+1
50 FOR X=0 TO 70
60 LOCATE X,Y:PRINT "▲"
70 FOR T=0 TO 80:NEXT
80 LOCATE X,Y:PRINT " "
90 NEXT
100 GOTO 40
Ok
```

アルファベットのキーを叩くと、つぎの行番号の命令から画面に表示していく

アルファベットのキーを任意に叩くと実行を再開

ん。

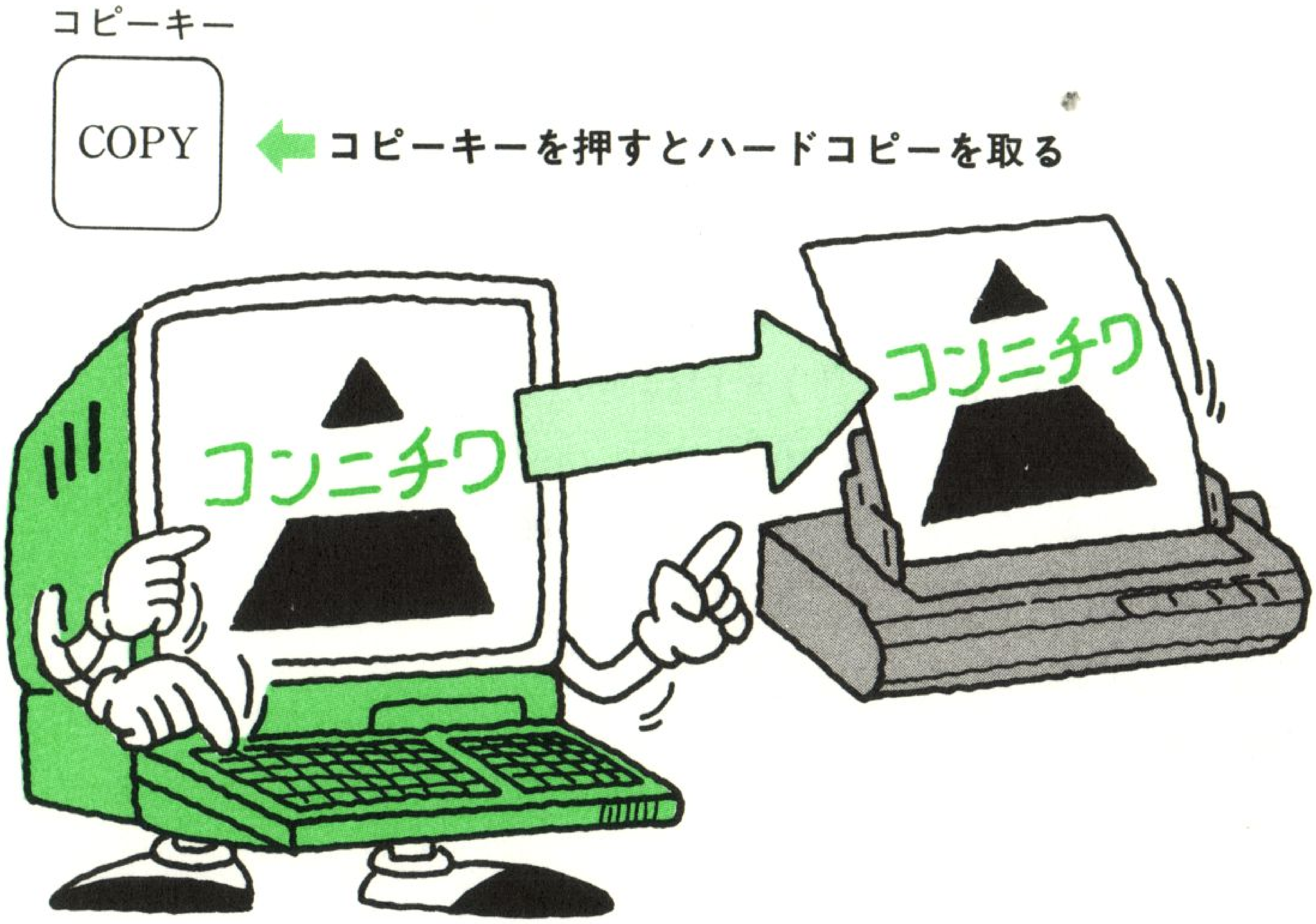
99頁に示したエンバンを飛ばすプログラムの場合も、CTRLキーを押しながらSキーを叩くとプログラムの実行が一時停止されて、エンバンの移動が止まります。

一時停止の再開

CTRLキーを押しながらSキーを叩いて実行を一時停止させた場合は、任意のアルファベットのキーを叩くと、ふたたび実行が開始されます。実行の開始は、実行が停止したつぎの命令からとなっています。

したがって、まえの頁に示した画面への表示の場合は、行番号50の命令のところで一時停止されましたから、上に示すように、行番号60の命令から画面に表示が再開されます。

なお、STOP、COPY、SHIFT、カナ、GRPHキーを叩いても、実行は再開されません。



[COPY] キーはハードコピーを取る

[COPY]キーは画面コピー

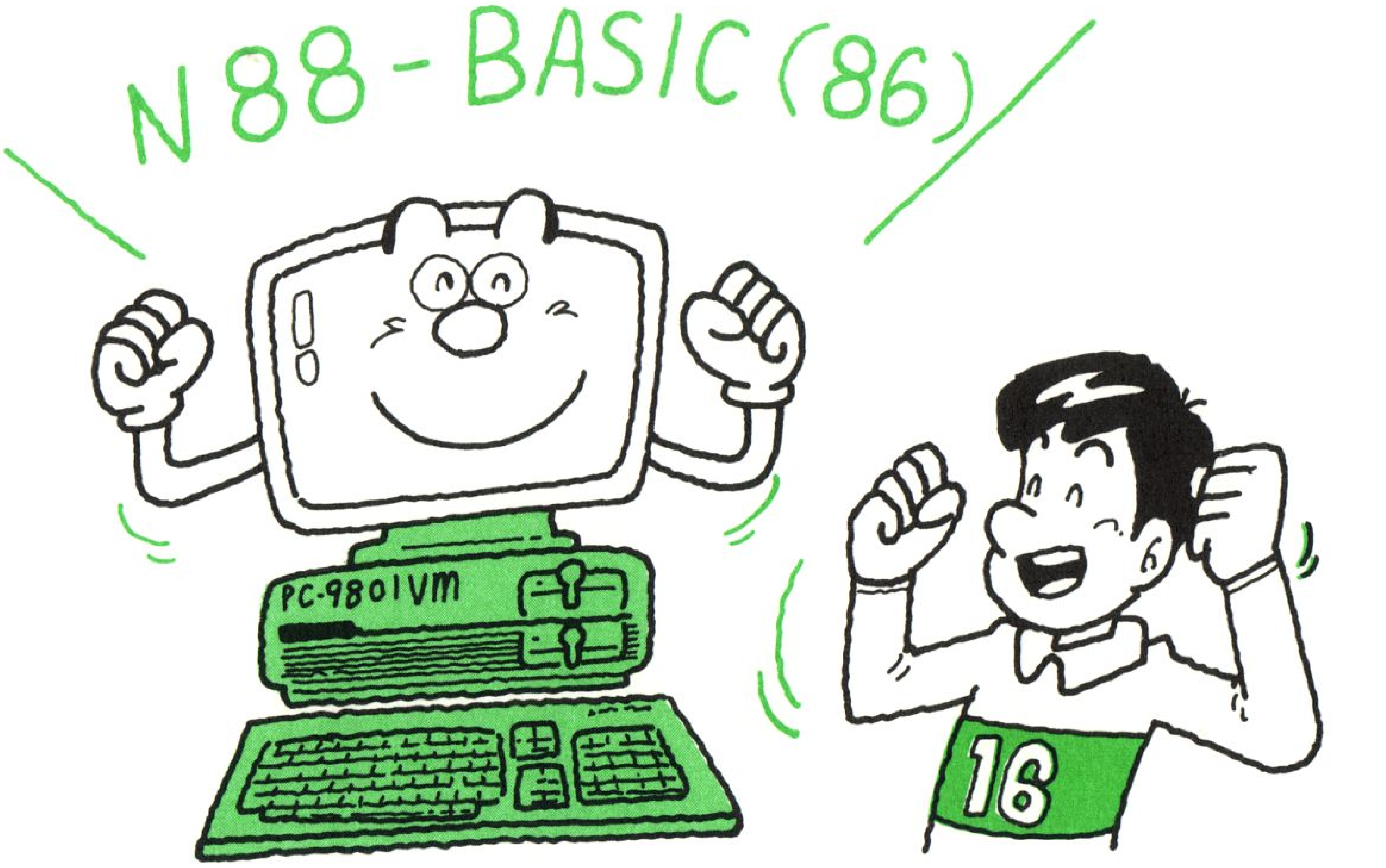
ディスプレイの画面に表示されている文字や図などを、画面に表示されているとおりにプリンタから印刷して取り出すことを、ハードコピーを取るといいます。

ハードコピーを取るには、[COPY]キーを押します。

[COPY]＝ハードコピーを取る

すると、上のイラストに示すように、画面に表示されている文字や図などがそのとおりにプリンタに出力されて印刷されます。

ハードコピーの場合も中止するときは、[STOP]キーを押すと中止されます。ただし、[CTRL]と[S]キーを押しても、一時停止の命令は無視されます。



電源を入れるとN88-BASICが起動する

パソコンを動かしてみよう

ベーシックはやさしいプログラミング言語（プログラムを作る言葉）で、すぐにおぼえることができます。また、いくつかの基本的な命令をおぼえてしまえば、パソコンを自由に動かすことができます。

パソコンのモデル機種として、ここではPC－9801VM2を使い、最初は本体内部のROMに記憶されているN88－BASIC(86)で命令を与えて動かすことにします。

N88－BASIC(86)とは、62頁で示したベーシックインタプリタの名称です。

PC－9800シリーズの場合、ベーシックインタプリタはROMとディスクの両方に記憶されていて、ROMに記憶されているベーシックイン

タプリタをＲＯＭ　ＢＡＳＩＣ、ディスクに記憶されているベーシックインタプリタをＤＩＳＫ　ＢＡＳＩＣと呼んでいます。

ふつう私たちが使うベーシックの命令は、ＲＯＭ　ＢＡＳＩＣのほうに記憶されています。

なおＰＣ－９８００シリーズの機種は、過去に出た機種であっても、また今後、新しく改良されて出る機種であっても、ベーシックインタプリタがＮ88－ＢＡＳＩＣ（86）であれば、基本的にはなんら変わることはありません。

ROM BASICの起動

16ビット・パソコンに電源を入れると、68頁に示したようにメッセージが表示されて、このパソコンがＮ88－ＢＡＳＩＣ（86）で動くことが示されます。

また、メッセージの最後にＯＫが表示されて、ベーシックの命令を入力してもよいということを示します。

これでＲＯＭ　ＢＡＳＩＣが起動したことになります。ＤＩＳＫ　ＢＡＳＩＣの起動のさせ方はディスクをディスク装置に挿入して行う関係から、192頁で取りあげます。

さて、[HOME CLR]キーでメッセージを消去したら、さっそくベーシックでパソコンを動かしてみることにします。

2×2の結果を画面に表示させる

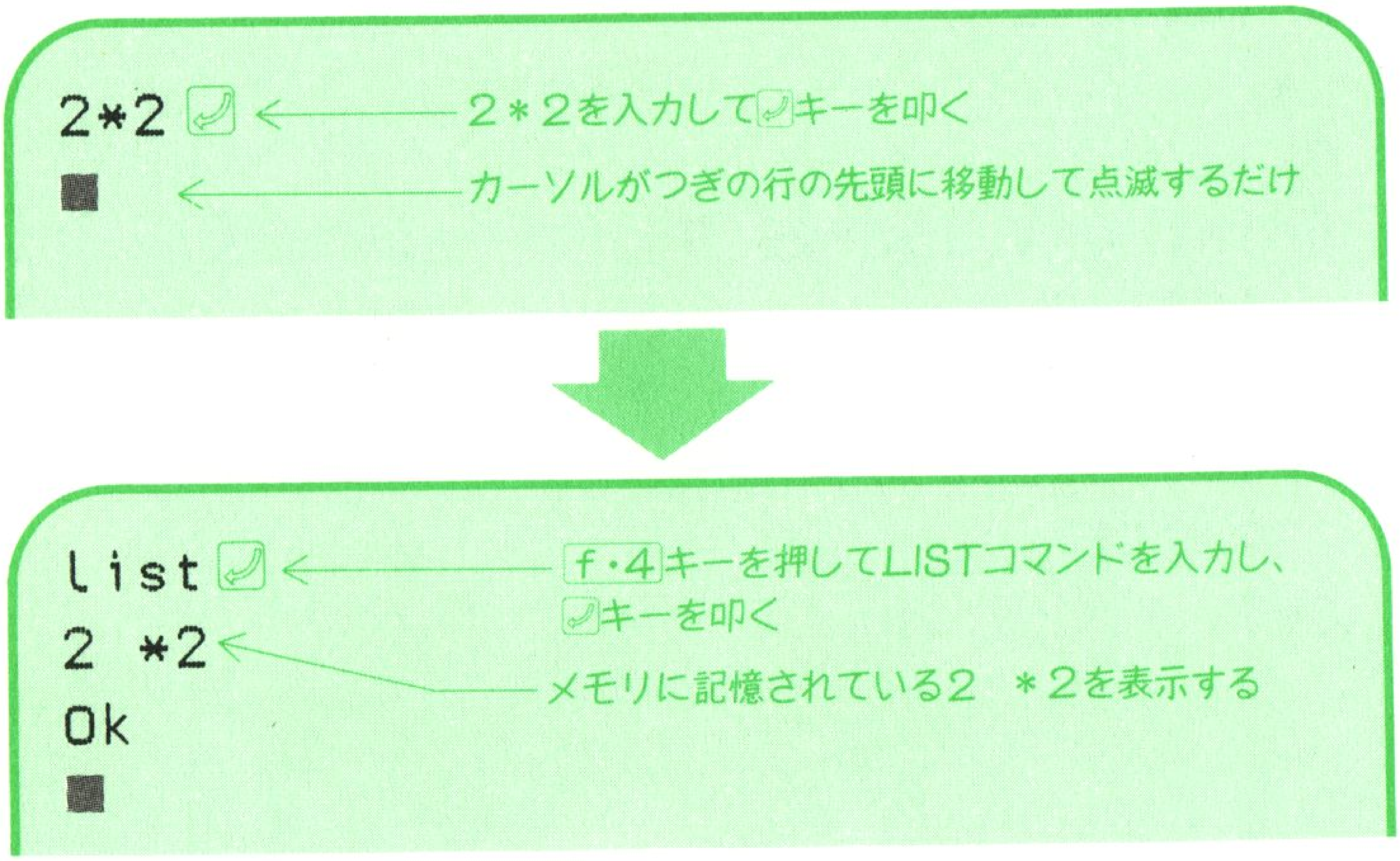
まず手初めに、２×２を行わせてみます。

2×2＝?

さて、[2]のキーはテンキーにありますが、×の記号はキーボードのどこを捜しても見当たりません。これは、×の記号は英文字のＸと間違いやすいので、＊（アスタリスク）の記号を使うようになっているからです。

したがって、２×２は、つぎのように入力することになります。

2＊2　↑ 2×2



2＊2だけを入力して ↵ キーを叩くと

もちろん、2＊2だけをキーボードのキーを叩いて入力しても、ベーシックの命令になっていないので、2＊2は計算されません。

ちなみに2＊2を入力して、↵キーを叩いてみると、上に示すように、ただカーソルがつぎの行の先頭に移動して点滅するだけで、2＊2の結果は表示されません。

では、入力した2＊2はどうなったのかというと、メモリに記憶されています。

ファンクションキーのf・4を押し、LISTコマンドを入力して↵キーを叩くと、メモリに記憶されている2＊2が、上に示すように画面に表示されます。

2＊2がメモリに記憶されたのは、入力された2＊2という形はベーシックの命令にないので、パソコンがどう処理してよいのかわからず、2を行番号、＊2をベーシックの命令と見なして、メモリに記憶してしまったわけです。

LISTコマンドで表示させた2と＊2の間が1文字分あいているのがその証拠です。

```
print 2*2 ⏎      ← ⏎キーを叩く
 4               ← 結果を表示
Ok
print 444.152*25.9⏎   ← ⏎キーを叩く
 11503.5              ← 結果を表示
Ok
print 3.210*9.1⏎      ← ⏎キーを叩く
 29.211               ← 結果を表示
Ok
```

どんなに複雑な掛け算でも正確に結果を表示

PRINTは画面に表示

2＊2の結果を画面に表示させるには、PRINT命令を、つぎのように2＊2のまえにつけます。

PRINT 2＊2 ←2×2の結果を画面に表示

これでベーシックの命令になりましたから、これを入力します。PRINTは、[P][R][I][N][T]と1文字ずつキー入力してもよいのですが、間違いがおきないように、[f・8]キーを押して入力します。

入力し終わったら⏎キーを叩きます。⏎キーを叩かないと、入力した命令は画面に表示されているだけで、実行されません。

⏎キーが叩かれると、パソコンは、さきのような間違いをおこすことなく即座に4という結果を、上に示すように画面に表示します。

どのように複雑な掛け算でも、計算する数さえ指定すれば、即座に、しかも正確に結果を画面に表示します。

```
print 10+ ↵          ← 入力間違い
Missing operand      ← エラーメッセージ
Ok
print 10+10 ↵        ← 新たに命令を入力して↵キーを叩く
 20
Ok
print *30 ↵          ← 入力間違い
Syntax error         ← エラーメッセージ
Ok
print 50*30 ↵        ← 新たに命令を入力して↵キーを叩く
 1500
Ok
```

ダイレクトモードで入力を間違えたときは新たに入力する

このように、ベーシックの命令を入力して↵キーを叩くと、それを即座にベーシックインタプリタでCPUが理解できるマシン語に変換して実行する状態を、**ダイレクトモード**といいます。

命令の入力を間違えたときは

ダイレクトモードで実行させることができるベーシックの命令は、掛け算の例で示したように1行だけですから、もしエラーメッセージが表示された場合は、新たにベーシックの命令を入力しなおして、↵キーを叩きます。

なぜなら、96頁で示したように、カーソルキーでカーソルを訂正する文字の上に移動して、正しい文字を入力するなどということをやっているよりも、新たにベーシックの命令を入力しなおしたほうが手っ取り早いからです。

なお、上に示した小数点をともなう数字の小数点は、テンキーの一番下の段、↵キーの左隣りにある・キーを押すと入力することができます。

100＋100
245－124
123＊23
4235／67

演算子が１個の式

220－20＊30
100＊20＋234／18
234／22－435＊3
(100＋50)＊3

演算子が複数個の式



演算子と式

四則演算と演算子

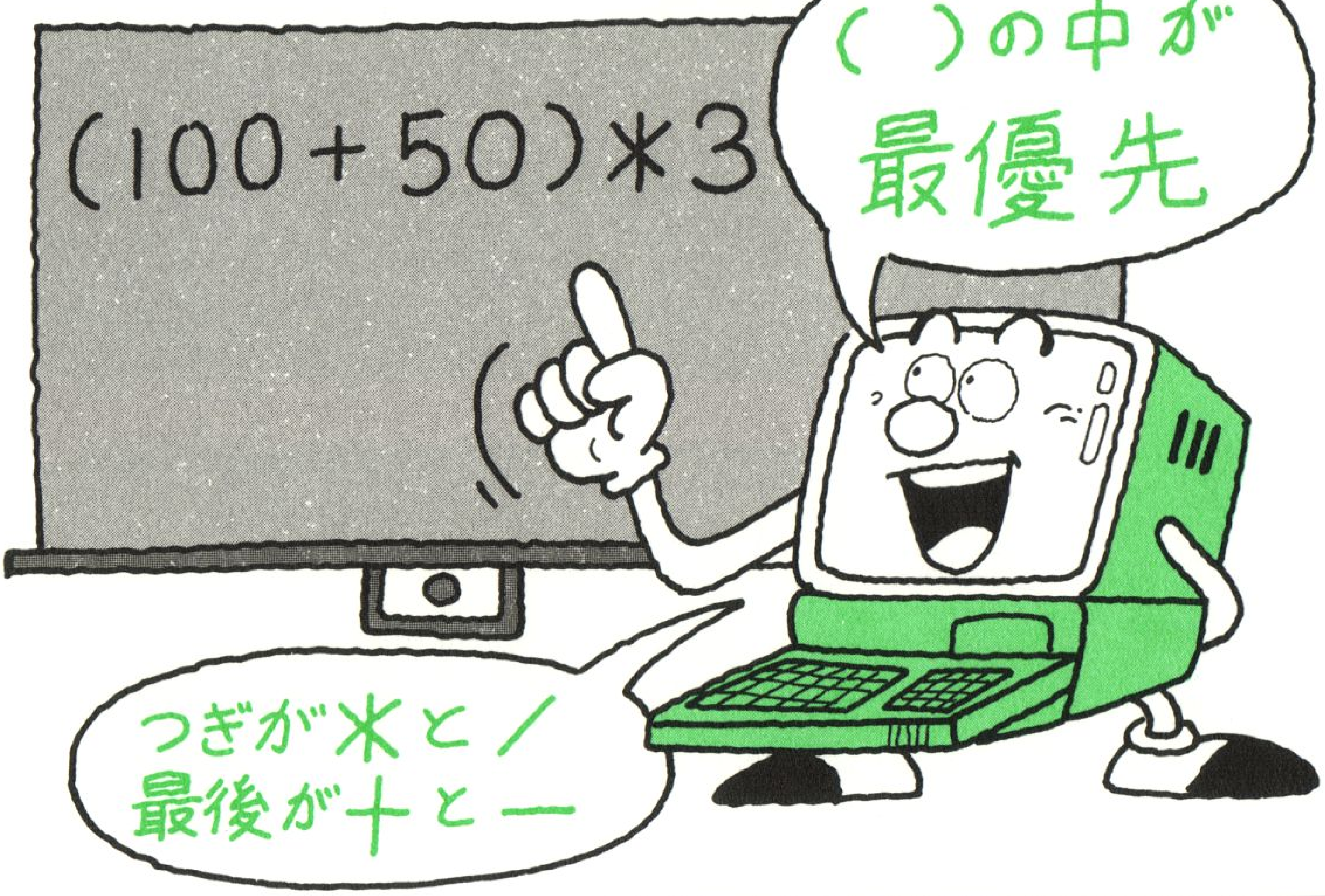
四則演算とは、足し算、引き算、掛け算、割り算のことです。この四則演算で使う記号のことを**演算子**といいます。

演算子は、足し算と引き算の二つは、私たちがいつも使っている記号とおなじで＋、－ですが、掛け算と割り算の記号は×、÷は使わずに、×は＊（アスタリスク）、÷は／（スラッシュ）を使うことになっています。

＋　**足し算（加算）**
－　**引き算（減算）**
＊　**掛け算（乗算）**
／　**割り算（除算）**

これらの演算子を使って、上に示すように、数字を結んだ数式を式といいます。

この式で、一つの式に複数個の演算子を使うときは、演算子に決められている順序にしたがって計算を行いますから注意が必要です。



計算の順序

演算子の計算順序

演算子の計算の順序は、*や/が+や−より先に計算が行われ、*や/の計算が行われたあとで+や−の計算が行われます。

*・/　→　+や−より先に計算される

+・−　→　*や/のあとで計算が行われる

したがって、つぎのような式の場合は、

```
100−20*6
    ↑  ↑
    ②  ①←計算の順序

① 20*6=120

② 100−120=−20
```

*が−より先に計算が行われますから、計算の順序は①、②となり、20*6が先に計算されて、120が求められます。

つぎに−の計算が行われますから、20*6の結果の120を100から引くことになります。

```
        ②  ①
print 100-20*6 ⏎
-20          ←―――――― 計算結果
Ok
```

```
        ①  ③  ②
print 100*2+60/30 ⏎
 202         ←―――――― 計算結果
Ok
```

演算子の計算順序と計算結果

100＊2＋60／30

↑　↑　↑

①　③　②←計算順序

① 100＊2＝200

② 60／30＝2

③ 200＋2＝202

　また、上のような場合は、＊と／が＋より先に計算が行われます。ただし、＊と／は、計算の順序がおなじです。一つの式におなじ計算順序の演算子がある場合は、左から右へという順序で計算が行われますから、この場合は／より左にある＊が先に行われ、そのあとで／が行われることになります。＋や－の場合もおなじです。

　したがって100＊2＋60／30の計算の順序は①、②、③となりますから、まず①の掛け算の100＊2＝200が行われます。

　つぎに②の割り算の60／30が行われます。

　そのあと100＊2の結果の200と、60／30の結果の2とが足されて、202が計算結果となります。

　参考までに、100－20＊6と100＊2＋60／30の実行結果を、上に示します。

```
         ①      ②
print (400+200)/5 ⏎   ← 400に200を足した結果を5で割る
 120                  ← 計算結果
Ok
```

```
          ②    ①
print 400+200/5 ⏎     ← 200を5で割った結果を400に足す
 440                  ← 計算結果
Ok
```

カッコでくくった場合の結果の違い

カッコのなかが最優先

式のなかにカッコでくくられた演算子があると、カッコでくくられた演算子が一番最初に計算が行われます。

たとえ、カッコのなかの演算子が＋や－であっても最優先されます。

(100＋50)＊3
　　↑　　↑
　　①　　②

① 100＋50＝150

② 150＊3＝450

たとえば、上の（100＋50）＊3のような場合は、カッコのなかの100＋50が先に計算されます。

そのあと、その結果の150に3が掛けられて、計算結果の450が求められます。

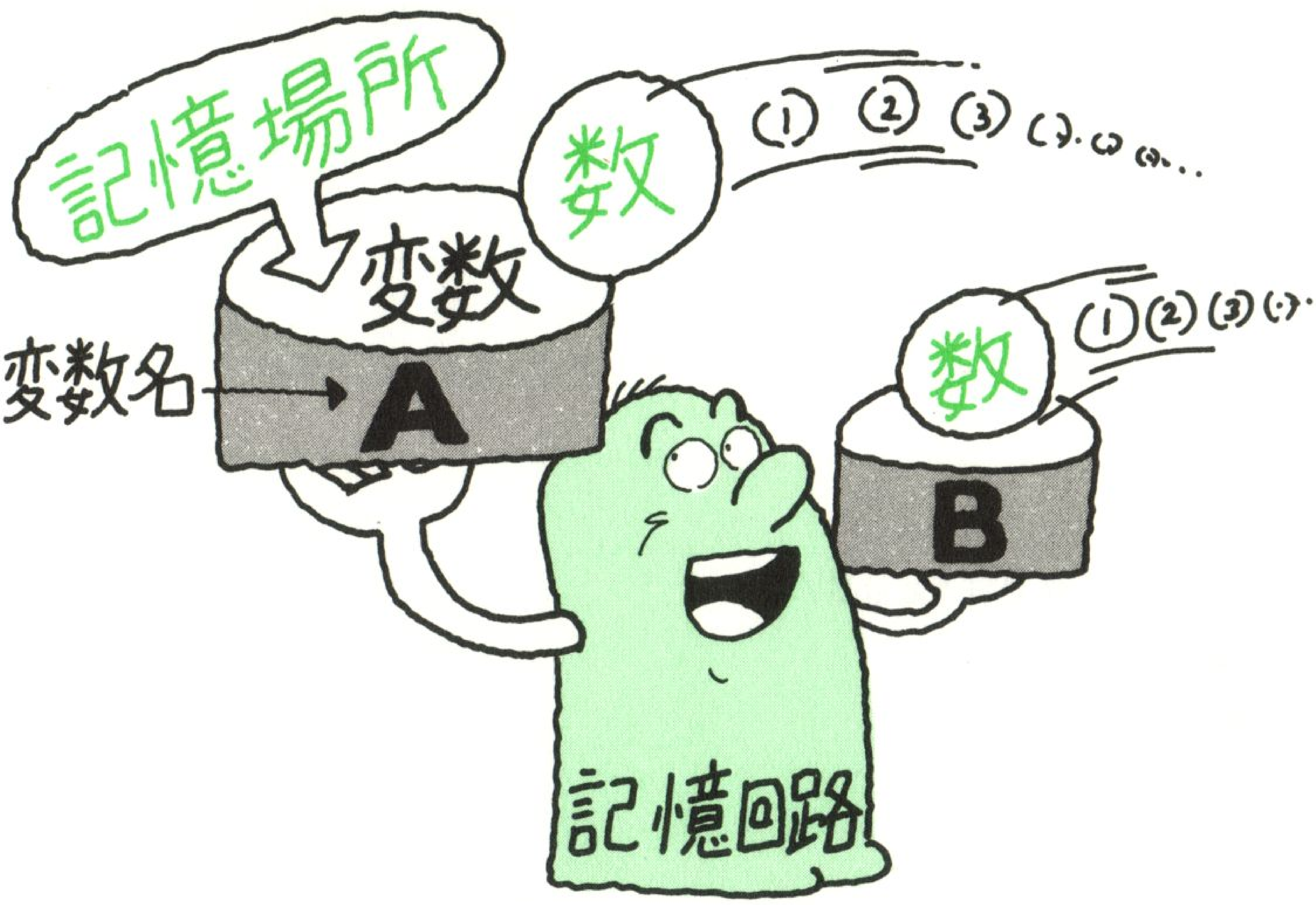
したがって、たとえば400に200を足し、それを5で割るというとき、

400＋200／5

とするのは間違いで、つぎのようにします。

（400＋200）／5

参考までに、この二つの計算結果がどのように違うかを、上に示します。



変数は数を記憶する場所の名前

変数は記憶場所の名前

さきの場合は、直接PRINTのあとに数を指定して計算しましたが、つぎは記憶場所に数を記憶させたうえで、計算してみることにします。

ダイレクトモードで計算を行う場合、直接PRINTのあとに数を指定して計算するのがふつうで、記憶場所に数を記憶させたうえで計算するなどという面倒なことはしません。

記憶場所に数を記憶させてから計算するのは、このあとで取りあげるプログラムモードですが、プログラムモードの説明のまえに、記憶場所に数を記憶させる方法を取りあげるのは、そのほうがわかりやすいからです。

さて、39頁で示したように、記憶回路のなかはいくつもの記憶場所に細かくわけられています。

ベーシックの場合、この記憶場所に数を記憶させるには、アルファベットを使って記憶場所を指定します。

A	B	C	
100	30	A＋Bの 結果 130	

変数への数の記憶の仕方

たとえば、Ａと指定すると、いくつもある記憶場所のなかからＡのための記憶場所を一つ選び出して、Ａという名前をつけます。そしてＡに記憶させる数が入力されると、そこに記憶します。Ｂと指定すると、おなじように記憶場所にＢという名前をつけ、Ｂに記憶させる数が入力されると、そこに記憶します。

このように、ＡあるいはＢという名前がつけられた記憶場所のことを**変数**、記憶場所につけたＡあるいはＢという名前のことを**変数名**といいます。

また、ＡあるいはＢという名前のついた記憶場所のことを変数Ａ、変数Ｂと呼びます。

変数Ａ、変数Ｂなどに数を記憶させるには、＝記号を使います。たとえば、変数Ａに100を記憶させるとすれば変数Ａと100を＝で結びます。

A＝100 ⏎　←　変数Ａに100を記憶させる

これを、キーボードのキーを叩いて入力し、⏎キーを叩くと、さきに示したように記憶場所にＡという名前つけて100を記憶します。

記憶されているかどうかは、ＰＲＩＮＴのあと

```
a=100 ↵      ←――――― 変数Aに100を代入
Ok
print a ↵    ←――――― 変数Aの中身を表示させる
 100         ←――――― 記憶されている100
Ok
```

```
b=30 ↵       ←――――― 変数Bに30を代入
Ok
print b ↵    ←――――― 変数Bの中身を表示させる
 30          ←――――― 記憶されている30
Ok
```

変数に値を代入する

に変数Aを指定します。

PRINT A ↵ ↑ 変数Aの中身を表示させる

これをキーボードのキーを叩いて入力し、↵キーを叩くと、上に示すように、変数Aに記憶されている100が表示されます。

また、変数Bに30を記憶させるとすれば、おなじように変数Bと30を=で結びます。

B=30 ↵ ↑ 変数Bに30を記憶させる

これを入力し、↵キーを叩くと変数Bに30が記憶されます。

この場合も、PRINTのあとに変数Bを指定して↵キーを叩くと、上に示すように変数Bに記憶されている30が表示されます。

このように=で結んで、数を変数に記憶させることを、変数に値（数）を代入するといいます。

変数に記憶させた値の計算

変数に代入した値の計算は、さきにPRINT命令のあとに直接値を指定して、足したり引いた

```
a=100     ←――― 変数Aに100を代入
Ok
b=30      ←――― 変数Bに30を代入
Ok
print a+b ←――― A＋Bを実行させる
 130      ←――― A＋Bの結果
Ok
print a*b ←――― A×Bを実行させる
 3000     ←――― A×Bの結果
Ok
```

変数に記憶させた値の計算

り、掛けたり割ったりした場合とおなじです。また、計算の順序もおなじです。

A＋B ↑ 変数Ａの値と変数Ｂの値を足す
A－B ↑ 変数Ａの値から変数Ｂの値を引く
A＊B ↑ 変数Ａの値に変数Ｂの値を掛ける
A／B ↑ 変数Ａの値を変数Ｂの値で割る
(A＋B)／C ↑ Ａの値とＢの値を足し、Ｃの値で割る

さて、さきに変数Ａに100、変数Ｂに30を代入しました。このように、いったん変数に値を代入すると、新たに値を代入しないかぎり変数の値は変わりませんから、変数Ａの100、変数Ｂの30で、いろいろな計算を行うことができます。

たとえば、Ａ＋Ｂとすると、

PRINT　A＋B

100＋30が行われて、上に示すように、その結果の130が表示されます。

またＡ＊Ｂとすると、

PRINT　A＊B

100＊30が行われて、上に示すように、その結果の３０００が表示されます。

```
print (a-b)/b
 2.33333
Ok
```

← (a−b)/bを実行させる
← 計算結果

```
print a-a/b
 96.6667
Ok
```

← a−a/bを実行させる
← 計算結果

変数に記憶させた値の計算（つづき）

PRINT　(A−B)／B
　　　　　①　②

① 100−30＝70
② 70÷30＝2.33333

PRINT　A−A／B
　　　　　②　①

① 100÷30＝3.33333
② 100−3.33333
　＝96.6667

また（A−B）／Bの場合、まず100−30が行われ、つぎに70÷30が行われて結果が表示され、A−A／Bの場合も、それぞれ計算され、上に示すように表示されます。

このように変数に値を記憶させた場合は、その値をいろいろな計算に使うことができます。

計算の結果を記憶場所に記憶する

さてこれまでは、PRINT命令で即座に計算結果を画面に表示させました。

ふつうプログラムでは、このようにPRINT命令で即座に計算結果を画面に表示させないで、いったん記憶場所に記憶させたうえで画面に表示

```
a=230 [↵]      ←── 変数Aに230を代入する
Ok
b=340 [↵]      ←── 変数Bに340を代入する
Ok
c=a+b [↵]      ←── A＋Bを行って、その結果を変数Cに代入する
Ok
print c [↵]    ←── 変数Cに代入した計算結果を表示させる
 570           ←── 変数Cに代入されている計算結果
Ok
```

計算結果を記憶場所に記憶

させる方法をとります。

記憶場所に計算結果を記憶させるには、計算結果を記憶させる変数を指定して、その変数と計算式を＝で結びます。

たとえば、A＋Bの計算の結果を、変数Cに記憶させるとするなら、つぎのように変数CとA＋Bという計算式を＝で結びます。

C＝A＋B　↑ A＋Bの結果を変数Cに代入

このようにすると、A＋Bの計算の結果が、変数Cに代入されて記憶されます。

ここでは、変数Aに230、変数Bに340を代入して、A＋Bの結果を変数Cに代入してみます。

まず、上に示すように、A＝230と入力して[↵]キーを叩きます。

A＝230[↵]　↑ 変数Aに230を代入する

これで変数Aに230を代入しましたから、つぎに、B＝340と入力して[↵]キーを叩きます。

B＝340[↵]　↑ 変数Bに340を代入する

つぎに、C＝A＋Bと入力して[↵]キーを叩くと、

C＝A＋B[↵]　↑ A＋Bの結果を変数Cに代入

ABCDEFGHIJKLMNOPQRST
UVWXYZ1234567890ABCD　}　変数名が40文字でつけられているので正しい

ABCDEFGHIJKLMNOPQRST
UVWXYZ1234567890ABCDE　}　変数名が41文字でつけられているので間違い

K12345 ← 変数名の最初がアルファベットなので正しい

1GOKEI ← 変数名の最初が数字ではじまっているので間違い

GOKEI＊ ← 変数名に＊記号が使われているので間違い

TOKUTEN. 1 ← 変数名に「.」記号は使ってもよいので正しい

変数名のつけ方

A＋B、つまり230＋340が行われ、その結果の570が、変数Cに代入されます。

変数Cに代入した計算結果を表示させるには、

PRINT C⏎ ← 変数Cの中身を表示

と入力して⏎キーを叩くと、まえの頁に示すように、変数Cに代入された計算結果の570が表示されます。

変数名のつけ方

これまでの計算で示した変数名は、AとかB、あるいはCというように、アルファベット1文字でしたが、変数名は最大40文字までつけることができます。いいかえれば、変数名は40文字以内でつければよいということです。

変数名＝40文字以内でつける

変数名をつけるときに使える文字は、アルファベットと数字の二種類で、カタカナや記号は使うことができません。ただし記号のうち、「・」記号は使ってもよいことになっています。

```
print 12345 ⏎      ←―― 入力して⏎キーを叩くと
 12345             ←―― 12345が画面に表示される
Ok
```

```
print ﾓｸﾞﾗ ﾀﾀｷ ｹﾞｰﾑ ⏎   ←―― 入力して⏎キーを叩いても
                    ←―― なにも表示されない
Ok
```

```
print abcdefg ⏎    ←―― 入力して⏎キーを叩くと
 0                 ←―― 0が表示される
Ok
```

文字の表示(1)

変数名＝アルファベットと数字でつける

このように変数名はアルファベットと数字でつけますが、変数名の最初の1文字は、まえの頁の例に示すように数字ではなく、必ずアルファベットでなければなりません。

変数名の最初の文字＝アルファベット

カタカナなどの文字の表示

これまで、パソコンに簡単な計算を行わせてきました。

つぎは、カタカナやアルファベットなどの文字を画面に表示させることにします。

ところで、数値を画面に表示させるときは、つぎのように入力して⏎キーを叩くと、

PRINT 12345 ⏎

上に示すように、画面に12345が表示されますが、カタカナやアルファベットの場合は、このようなわけにはいきません。

たとえば、つぎのように入力して、

```
print "モグ゛ラ タタキ ゲ゛ーム"⏎ ←入力して⏎キーを叩くと
モグ゛ラ タタキ ゲ゛ーム ←カタカナが画面に表示される
Ok
```

```
print "ABCDEFG"⏎ ←入力して⏎キーを叩くと
ABCDEFG ←アルファベットが表示される
Ok
```

```
print "+/abc¥3"⏎ ←入力して⏎キーを叩くと
+/abc¥3 ←文字として表示される
Ok
```

文字の表示(2)

PRINT モグラ タタキ ゲーム

⏎キーを叩くと、まえの頁に示すように、画面にはなにも表示されません。

また、つぎのように入力して⏎キーを叩くと、

PRINT ABCDEFG

まえの頁に示すように、0が表示されます。

この場合、0が表示されるのは、「ABCDEFG」は変数と受けとられ、変数ABCDEFGには、なにも値を代入していないので、0が表示されるわけです。

「モグラ タタキ ゲーム」とか、「ABCDEFG」などのカタカナやアルファベットの文字を画面に表示するには、つぎのように、これらの文字を「"」引用符で囲みます。

PRINT "モグラ タタキ ゲーム"
PRINT "ABCDEFG"

これを入力して⏎キーを叩くと、上に示すように画面に表示されます。

「"」引用符号で囲んだ文字の並びを、**文字列**あるいは**ストリング**といいます。「"」引用符で

```
print "ABCDEFG"⏎   ←── 大文字を「〃」で囲むと
ABCDEFG            ←── 大文字で表示
Ok
```

```
print "abcdefg"⏎   ←── 小文字を「〃」で囲むと
abcdefg            ←── 小文字で表示
Ok
```

```
print "ABCDE  fghijkl"⏎   ←── 「〃」で囲んだ場合
ABCDE  fghijkl            ←── 大文字小文字を区別して表示
Ok
```

アルファベットは、大文字、小文字を区別して扱う

囲んだ文字が1文字であっても、文字列（ストリング）といいます。

また、「〃」引用符で囲んだ文字の並びが、つぎのように意味をなさないものであっても、

PRINT "+/abc¥3"

パソコンは、文字の並びを文字列（ストリング）として扱って、まえの頁の最後に示すように画面に表示します。

文字列は、大文字、小文字を区別する

ところで、アルファベットには大文字と小文字があります。画面にアルファベットを表示するために「〃」引用符で囲む場合、大文字で表示するときは、大文字で入力します。

PRINT "ABCDEFG"

すると、上に示すように大文字で表示されますが、小文字で入力すると、

PRINT "abcdefg"

小文字で表示されます。

```
print "12345"⏎
12345
Ok
```

←「"」で囲むと数値も文字として表示される

```
print "67890"+"12345"⏎
6789012345
Ok
```

←「"」で囲んだ数値を足すと
←連結して表示する

```
print "12345"+"ｱｲｳｴｵ"⏎
12345ｱｲｳｴｵ
Ok
```

←数値と文字を足すと
←数値と文字を連結して表示

「"」で囲むと数値も文字になる

数値も引用符で囲むと文字として扱われる

さて、数値も「"」引用符で囲むと、数値としての本来の意味を失って、文字列として扱われることになります。

PRINT "12345"

したがって、つぎのように「"」引用符で囲んだ数値を足すと、

PRINT "67890"+"12345"

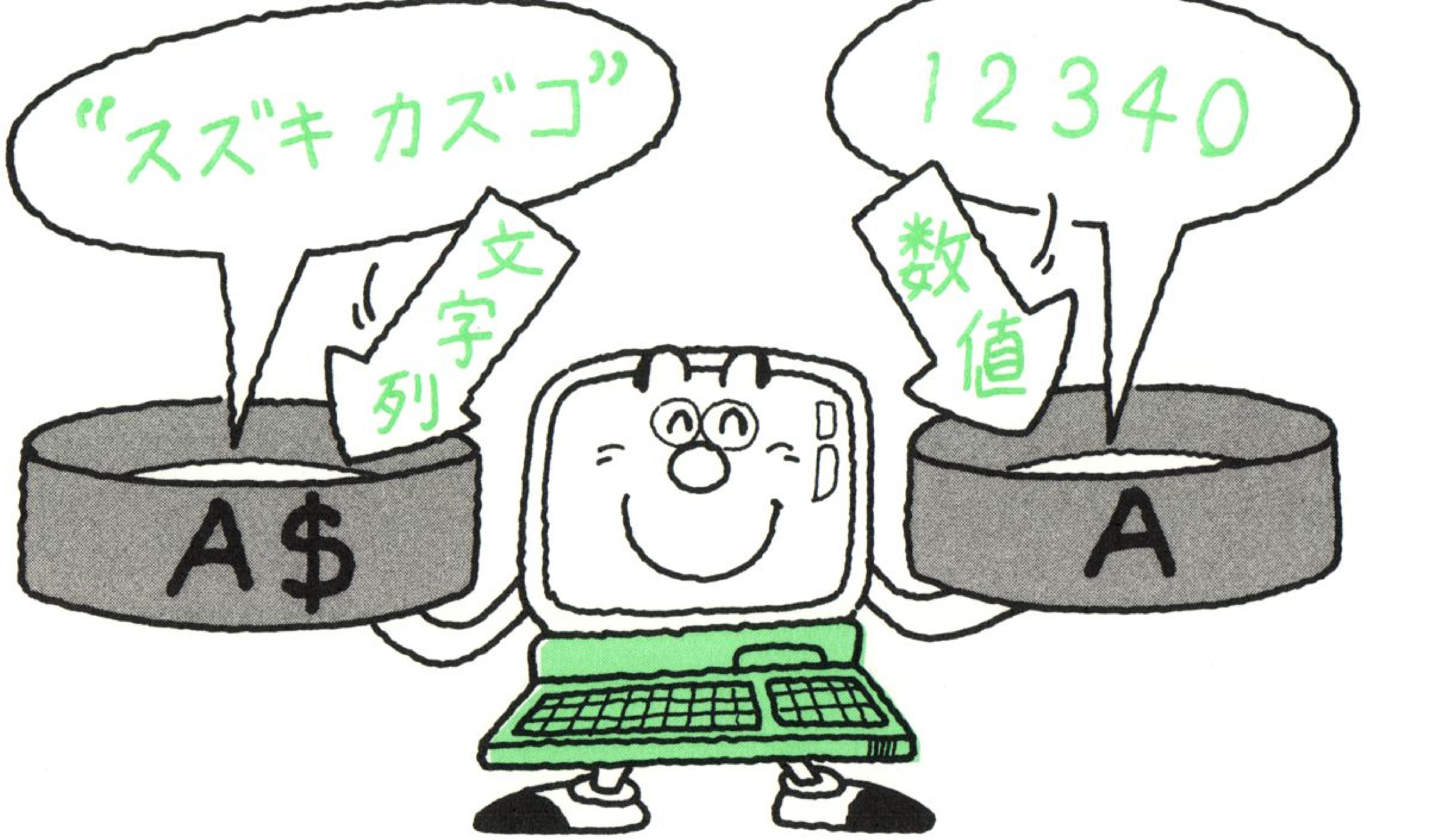
上に示すように、数値と数値が連結されて表示されることになります。

もちろん数値ばかりでなく、つぎのように数値とカタカナを足した場合は、

PRINT "12345"+"アイウエオ"

上に示すように、数値とカタカナが連結されて表示されます。

この場合の＋(プラス)記号は、足す記号ではなく、連結する記号であるところから、**連結記号**といいます。



文字変数と変数

文字変数は文字列を記憶する

この「〃」引用符で囲んだ文字列も、数値とおなじように記憶場所に記憶することができます。114頁で示したように、数値を記憶場所に記憶させる場合は変数を使いましたが、文字列を記憶させるときは、文字変数を使います。

文字変数は、つぎのように、変数のあとに「$」ダラーをつけたものです。

**A$　NAME$** ← 文字変数は変数のあとに$をつける

このように、文字変数は変数の最後に「$」をつければよいだけで、そのほかのことは120頁で示した変数名のつけ方とおなじです。

また、文字列を文字変数に記憶させる場合も、数値の場合とおなじように、文字列と文字変数とを「=」イコール記号で結べばよいのです。

たとえば、文字変数A$に〃スズキ　カズコ〃という文字列を記憶させるとすれば、A$と〃スズキ　カズコ〃とを「=」で結びます。

```
a$="スズキ カズコ"↵
Ok
```

← 文字変数A$に文字列を代入

```
print a$↵
スズキ カズコ
Ok
```

print a$↵ ← 文字変数A$の内容を表示させる

スズキ カズコ ← 文字変数A$の内容を表示

```
b$=saito kazuo↵
Type mismatch
Ok
```

← 引用符で囲まないとエラーになる

文字列と文字変数

A$＝"スズキ　カズコ"↵

　これを入力し、↵キーを叩くと、数値変数の場合とおなじように、記憶場所にA$をつけて、その記憶場所に"スズキ　カズコ"を記憶します。

　A$という記憶場所に"スズキ　カズコ"が記憶されているかどうかは、PRINTのあとにA$を指定します。

PRINT　A$↵

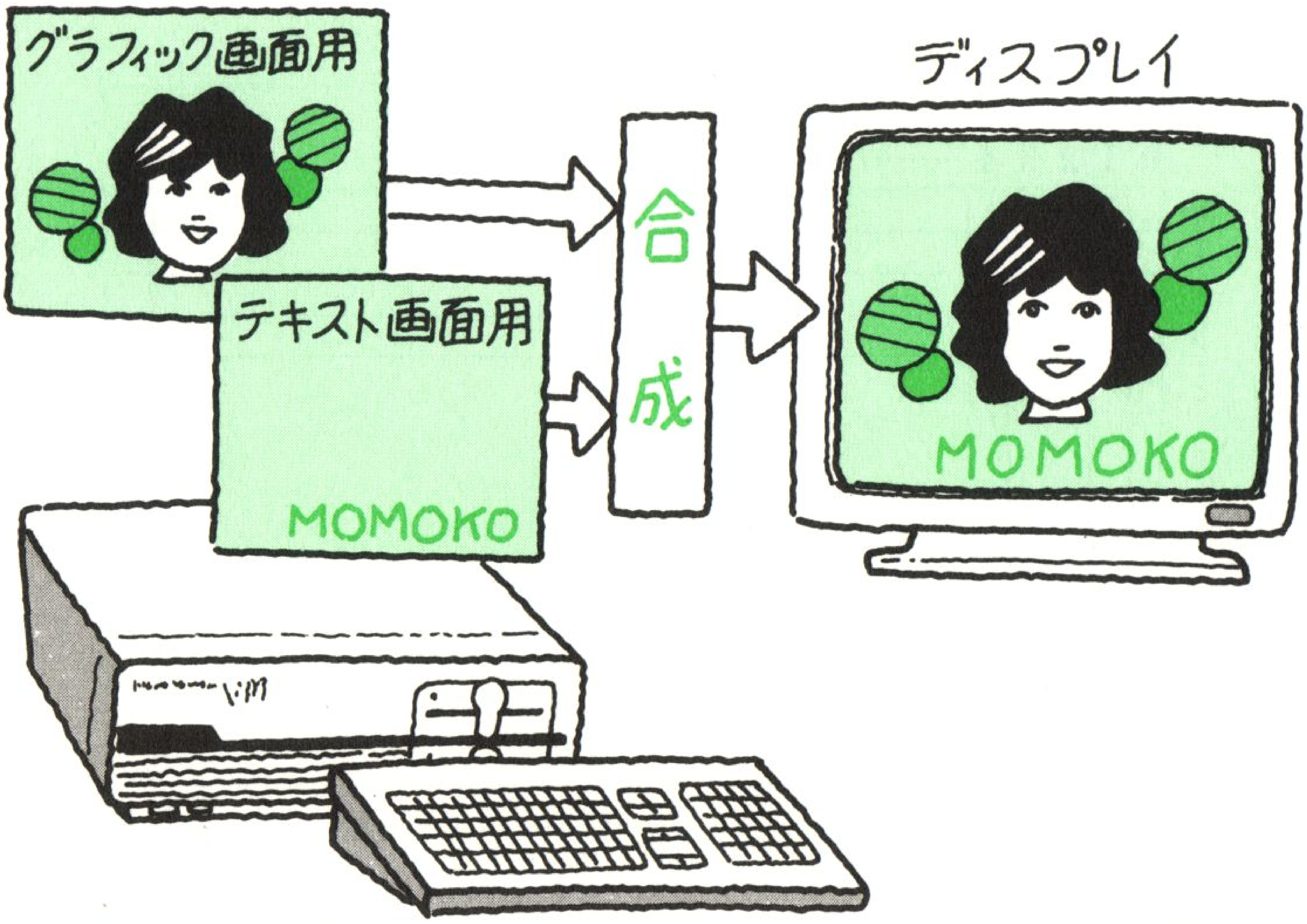
　これを入力して↵キーを叩くと、上に示すように、A$に記憶されている"スズキ　カズコ"が表示されます。

　文字変数に文字列を代入する場合、つぎのように「"」引用符で囲むのを忘れると、

A$＝スズキ　カズコ
N$＝SAITO
B$＝12345

← 「"」で囲まないと文字列ではないのでエラーになる

　文字変数に代入するものが文字列ではないので、上に示すように、エラーメッセージが表示されますから注意してください。



出力されるときにひとつに合成されて出力される

テキスト画面とグラフィック画面

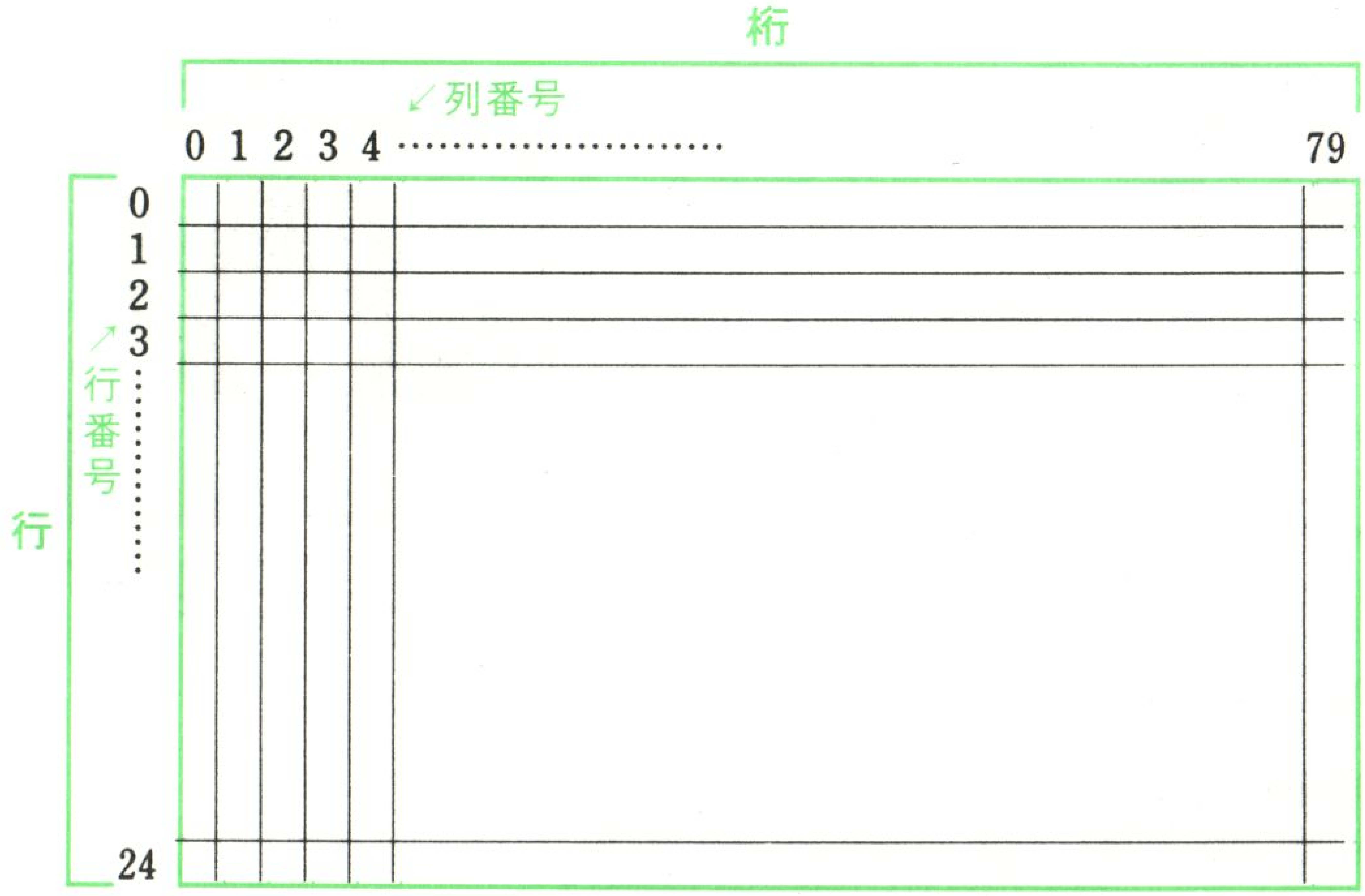
テキスト画面は文字を表示

どんなパソコンにも、テキスト画面とグラフィック画面の二種類があります。

テキスト画面とは、文字や数字、あるいは記号などを表示する画面のことです。グラフィック画面とは、点や線などを使ってグラフや図、あるいは絵などを表示する画面のことです。

テキスト画面は、これまで私たちが文字や数字、あるいは記号などを表示させてきた画面をいいます。

もちろんディスプレイの画面に、文字や数字などを表示させるテキスト画面と、グラフや図を表



テキスト画面の画面モード

示するグラフィック画面の、二枚の画面があるわけではありません。

ディスプレイの画面は、あくまで一枚なのですが、パソコンの本体内部で、文字などはテキスト画面用の記憶場所に記憶され、図などはグラフィック画面用の記憶場所に記憶されているということです。

このように、パソコンの内部で別々の記憶場所に記憶されている文字と図などが、ディスプレイの一枚の画面に同時に表示されるのは、ディスプレイの画面に出力するときに、文字と図などを一つに合成して出力するからです。

テキスト画面の画面モード

ところでPC－9801シリーズのテキスト画面は、横80、縦25のマス目に区切られています。もちろん、マス目で区切られているといっても、そのマス目は目で見ることはできません。

テキスト画面に表示される文字は、このマス目に1文字ずつ表示されていきます。

文字が表示されるマス目のうち、横方向のマス

目の並びを**桁**と呼び、また、縦方向のマス目の並びを**行**と呼びます。

そして、横方向のマス目の並びには、０から79までの番号がつけられ、この番号を**列番号**といいます。

また、縦方向のマス目の並びには、０から24までの番号がつけられ、この番号を**行番号**といいます。

このように、テキスト画面の横方向のマス目に列番号が、縦方向のマス目に行番号がつけられているのは、テキスト画面に表示する文字の位置を指定するためです。このことは、ＬＯＣＡＴＥ（ロケート）文（159頁）の項で示します。

さて、横80桁、縦25行のマス目がある場合、テキスト画面に表示することができる文字は、横80文字×縦25行ということになりますが、実際には画面の一番下にファンクションキーの内容が表示されているので、横80文字×縦24行ということになります。

ファンクションキーの内容の表示を消去すれば縦25行表示することができるので、画面の下にファンクションキーの内容が表示する行を含めて、横80文字×縦25行表示できるものとして、このテキスト画面を80桁25行モードといいます。

ファンクションキーの内容の表示の消去については134頁を参照してください。

ＰＣ－９８０１などのテキスト画面モードは、80桁25行モードばかりではなく、つぎのように４種類の画面モードがあります。

40桁20行モード ➡ 横40文字×縦20行
80桁20行モード ➡ 横80文字×縦20行
40桁25行モード ➡ 横40文字×縦25行
80桁25行モード ➡ 横80文字×縦25行

それぞれの画面モードに変更するには、ＷＩＤＴＨ（ウイドス）文を使います。

ＷＩＤＴＨ文とは、ＷＩＤＴＨ命令とおなじことですが、ベーシックの場合、命令とはいわずに文（ステートメント）といいますから、まえに取りあげたＰＲＩＮＴ命令は、正しくはＰＲＩＮＴ文となります。

画面モードを変更するWIDTH文の形

WIDTH 〈桁数〉, 〈行数〉

▲WIDTH文の〈桁数〉と〈行数〉のところには変更する画面モードの桁数、行数を指定する

WIDTH 40, 20	← 40桁20行モードに変更
WIDTH 80, 20	← 80桁20行モードに変更
WIDTH 40, 25	← 40桁25行モードに変更
WIDTH 80, 25	← 上記の画面モードに変更したあともとの80桁25行モードに戻す場合

WIDTH文の指定の仕方

WIDTH文は画面モードを変更

WIDTH（ウイドス）文は、さきに示した四つの画面モードに変更するために使うもので、上に示すような形をしています。

WIDTH文の〈桁数〉のところには、変更する画面モードの桁数を、〈行数〉のところには、変更する画面モードの行数を指定します。

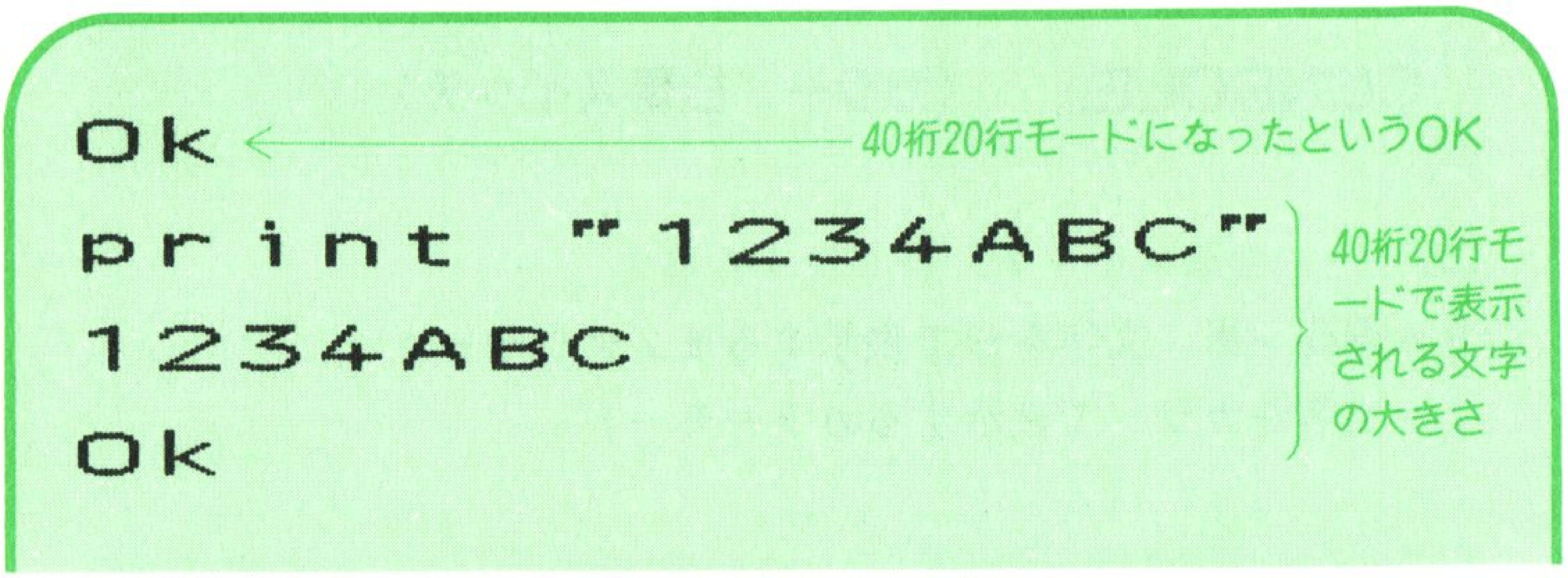
たとえば、80桁25行モードを40桁20行モードに変更するには、〈桁数〉のところには40、〈行数〉のところには20と指定します。

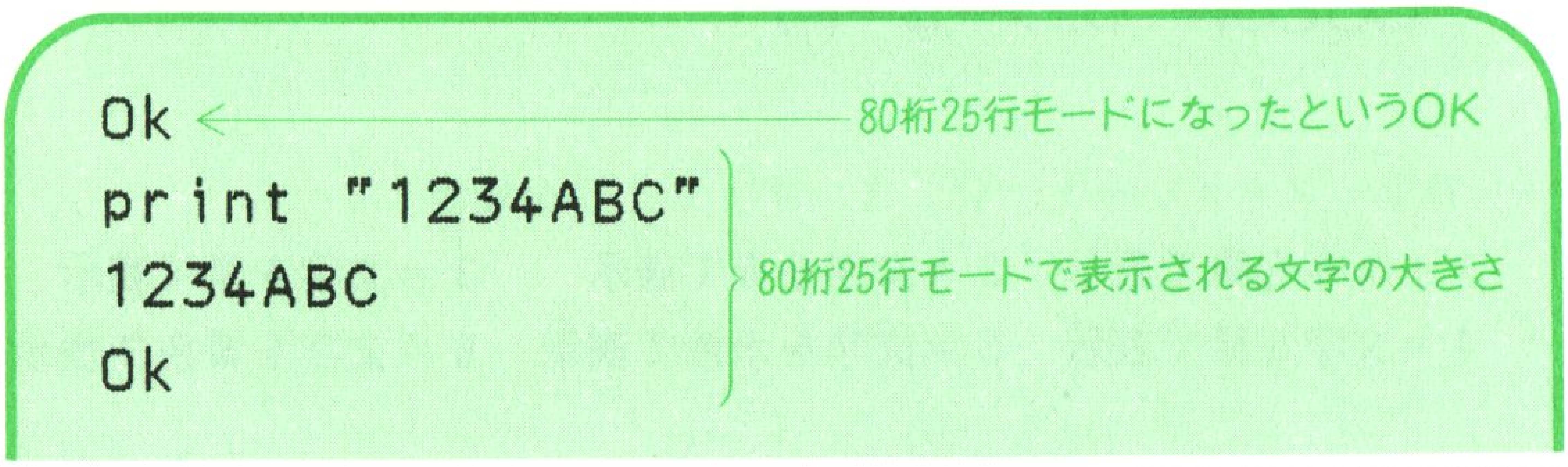
WIDTH 40,20⏎ ← 40桁20行モードに変更

この場合、40と20の間にあるコンマの入力を忘れると、Illegal function call というエラーメッセージが表示されます。

これを入力して⏎キーを叩くと、80桁25行モードが40桁20行モードに変更されます。

この場合、⏎キーを叩くと同時に、画面に表示されていたWIDTH文の命令の文字は消えて、





40桁20行モードと80桁25行モードの文字の大きさ

画面モードが変更されたというOKとカーソルが画面に表示されます。

40桁20行モードの画面に表示される文字の大きさは、画面が横40文字、縦20行表示になることから、上に示すように、80桁25行モードで表示される文字より、かなり大きな文字となります。

ところで、WIDTH文で画面モードを変更すると、ふたたびWIDTH文で画面モードを変更しないかぎり、その画面モードがいつまでもつづきます。

たとえば、画面モードを40桁20行モードにすると、いつまでも40桁20行モードがつづくことになります。

したがって、40桁20行モードを、もとの80桁25行モードに戻すときは、WIDTH文につぎのように指定して入力し、

WIDTH 80, 25 ⏎

⏎キーを叩いて、画面モードを80桁20行モードに変更しなければなりません。

CONSOLE,,, 〈カラー／白黒スイッチ〉

〈カラー／白黒スイッチ〉➡0か1を指定

0＝画面を黒、文字を白で表示するモノクロモード

1＝文字をカラーで表示するカラーモード

COLOR 〈カラーコード〉

〈カラーコード〉➡1、2、3、4、5、6を指定

1＝文字を青で表示	2＝文字を赤で表示	3＝文字を紫で表示
4＝文字を緑で表示	5＝文字を水色で表示	6＝文字を黄色で表示

CONSOLE文とCOLOR文

文字などをカラーで表示

ディスプレイがカラーディスプレイであっても、パソコンに電源を入れただけでは、画面の色は黒、文字や数字などは白で表示されて、カラーで表示させることはできません。

文字や数字などを画面にカラーで表示させるには、CONSOLE（コンソール）文とCOLOR（カラー）文の二つで命令します。

文字や数字などをカラーで表示させる場合のCONSOLE文とCOLOR文は、つぎに示すような形になります。

CONSOLE,,,〈カラー／白黒スイッチ〉

CONSOLE文でカラーモードを指定

文字や数字などをカラーで表示するには、まずCONSOLE文でカラーモードを指定します。

CONSOLE文には、4箇所指定するところがあります。

```
Ok                    ←文字などをカラーで表示するOK ┐
color 6               ←文字などを黄色で表示する指定 ┘ 白で表示
Ok                                                  ┐
print "12345ABCDEF"                                 │
12345ABCDEF                                         │ 文字などを黄色で表示
Ok                                                  │
color 3               ←表示する色を紫に変更          ┘
Ok                                                  ┐
print "アイウエオ*+=/"                               │ 文字などを紫で表示
アイウエオ*+=/"                                      │
Ok                                                  ┘
```

画面に表示する文字などの色の指定

まえに示したのは、文字をカラーで表示する指定のところだけです。〈カラー／白黒スイッチ〉のまえにある三個の「，」コンマは、3箇所の指定を省略した印ですから、三個の「，」コンマは忘れずに入力しなければなりません。

さて、CONSOLE文の〈カラー／白黒スイッチ〉のところに1を指定して、まずカラーモードにします。

CONSOLE，，，1⏎ ←カラーモードにする

これを入力して⏎キーを叩くと、CONSOLE文の文字は消えて、上に示すようにOKが表示されてカラーモードになります。

■COLOR文で文字の色を指定

カラーモードになったら、COLOR文の〈カラーコード〉のところに、1から6までを指定して、表示する文字の色を指定します。

たとえば、つぎのように、COLOR文の〈カラーコード〉のところに6を指定して入力し、

COLOR 6⏎ ← 文字などを黄色で表示

⏎キーを叩くと、このあとの文字や数字などは、

CONSOLE,, 〈ファンクションキー表示スイッチ〉〈カラー／白黒スイッチ〉

CONSOLE ,, 0 ←	ファンクションキーの内容の表示を消去する
CONSOLE ,, 1 ←	ファンクションキーの内容を表示する
CONSOLE ,, 0, 1 ←	ファンクションキーの内容の表示を消去して、カラーモードにする
CONSOLE ,, 1, 0 ←	ファンクションキーの内容を表示させて、モノクロモードにする

ファンクションキーの内容の表示の消去

黄色で表示されます。

黄色から紫に色を変えるときは、つぎのように〈カラーコード〉のところを3に指定して入力し、↵キーを叩くと、

COLOR 3 ↵ ← 文字を紫で表示

そのあとの文字や数字は、紫で表示されます。

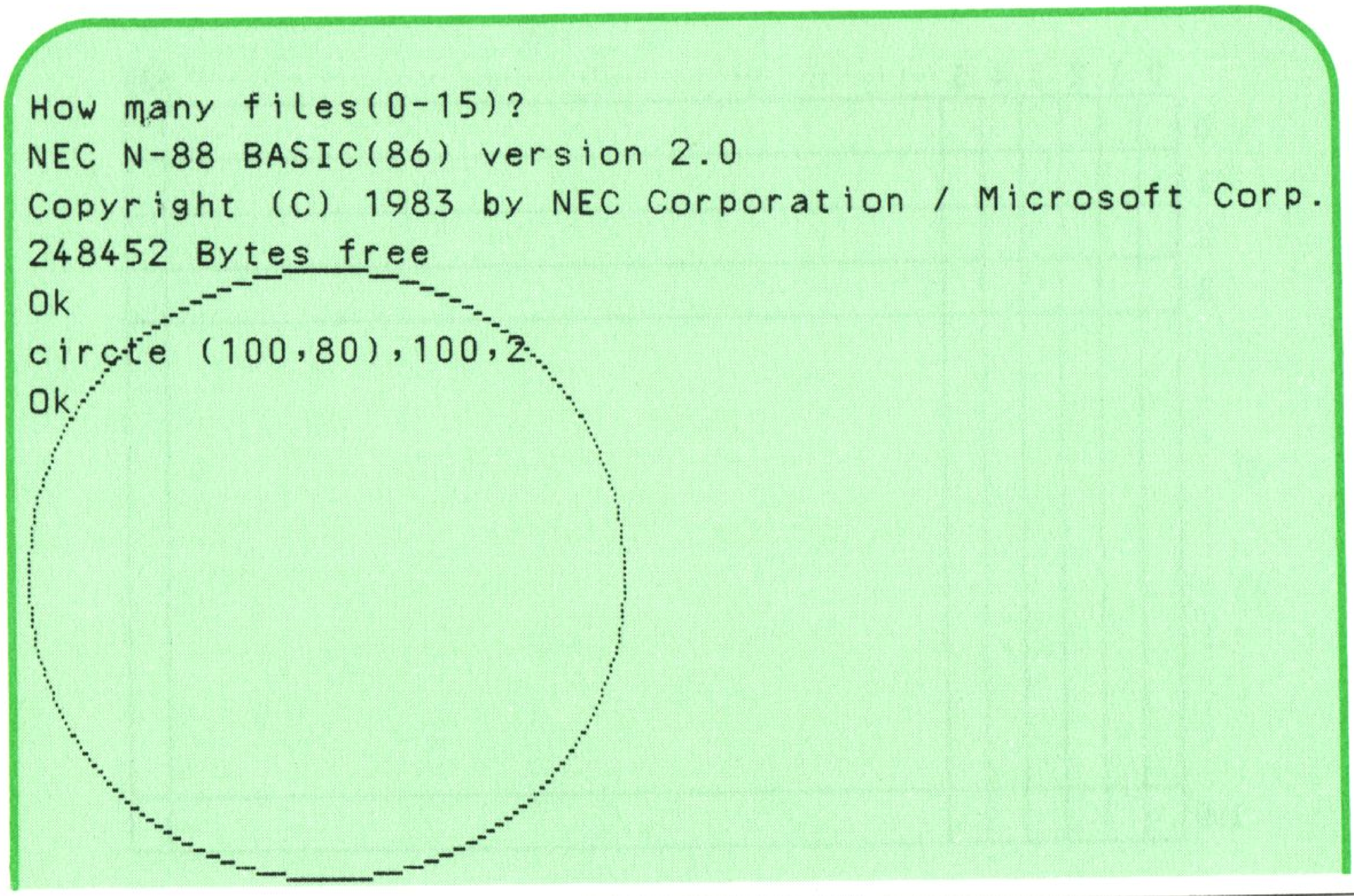
カラーモードからモノクロモードに戻すときは、CONSOLE文の〈カラー／白黒スイッチ〉のところを、つぎのように0と指定して入力し、

CONSOLE,,,0 ↵ ← モノクロモードに戻す

↵キーを叩けばよいのです。

なお、上に示すように、CONSOLE文の〈カラー／白黒スイッチ〉のまえは〈ファンクションキー表示スイッチ〉となっていて、このところを0と指定して入力し、↵キーを叩くと、画面の下に表示されているファンクションキーの内容の表示が消去されます。

内容を表示させるときは、〈ファンクションキー表示スイッチ〉のところを1と指定して入力し、↵キーを叩けばよいのです。



グラフィック画面とテキスト画面

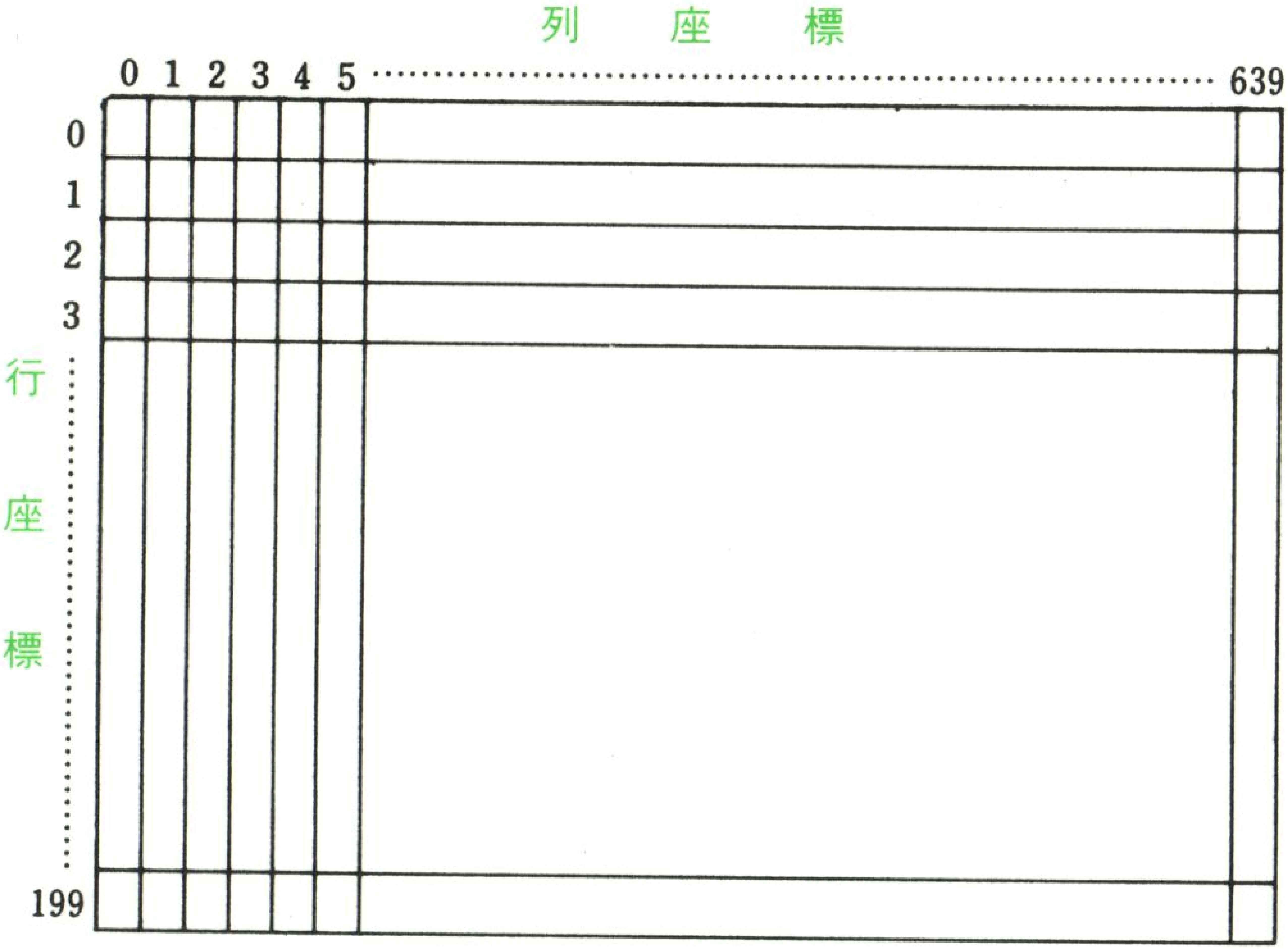
グラフィック画面には図を表示

127頁で示したように、パソコンには、テキスト画面に表示させる文字などを記憶しておく記憶場所と、グラフィック画面に表示させる図などを記憶しておく記憶場所とがあります。

このように、パソコン本体のなかで、グラフィック画面用の記憶場所とテキスト画面用の記憶場所とが分離されているので、上に示すように、ディスプレイの画面に多くの文字が表示されていても、それとは関係なく円や線を表示させることができます。

グラフィック画面の画面モード

テキスト画面の場合は、横80、縦25のマス目に区切られていて、そのマス目に一つ一つの文字を表示していきましたが、グラフィック画面の場合は、横640、縦200という、もっと細かいマス目に区切られていて、このマス目の一つ一つに点を表示していって、図などを描いていきます。ふつう、



横640×縦200ドットのグラフィック画面

点のことを**ドット**といいます。

また、テキスト画面の場合は、横方向を桁、縦方向を行といいましたが、グラフィック画面の場合は、横方向を**列座標**、縦方向を**行座標**といいます。そして、列座標には0から639までの番号がつけられ、行座標には0から199までの番号がつけられています。

この列座標の番号と行座標の番号を指定して、図などを描くわけです。

PC－9801などのグラフィック画面の画面モードには、つぎのように四種類あります。

カラーモード　横640ドット×縦200ドット
モノクロモード　横640ドット×縦200ドット
高分解能モノクロモード　横640ドット×縦400ドット
高分解能カラーモード　横640ドット×縦400ドット

高分解能の場合は、横640ドット、縦400ドットとなりますから、さらに細かいドットで図などを描くことができます。

SCREEN 〈画面モード〉, 〈画面スイッチ〉

〈画面モード〉➡0から3までを指定する

0＝横640×縦200ドットのカラーモード
1＝横640×縦200ドットのモノクロモード
2＝横640×縦400ドットの高分解能モノクロモード
3＝横640×縦400ドットの高分解能カラーモード

〈画面スイッチ〉➡0から3までを指定する

0か1＝描いた図がグラフィック画面に表示される
2か3＝描いた図がグラフィック画面に表示されない

グラフィック画面の画面モードの指定

SCREEN文で画面モードを指定

さきに示したグラフィック画面の画面モードはSCREEN（スクリーン）文で指定して選びます。

SCREEN文には、上に示すように、指定するところが二つあります。

〈画面モード〉のところに0から3までを指定して、四種類の画面モードのうちのいずれかの画面モードを選びます。

〈画面スイッチ〉のところにも0から3までを指定しますが、0か1を指定すると、描いた図がグラフィック画面に表示されます。

ここでは、横640ドット×縦200ドットのカラーモードで円を描いてみることにします。

したがって、SCREEN文の〈画面モード〉のところには0を指定します。

また、描いた円は画面に表示しますから、SCREEN文の〈画面スイッチ〉のところに0か1のいずれを指定します。

```
SCREEN 0, 0
```

CIRCLE (X, Y), 〈半径〉, 〈パレット番号〉

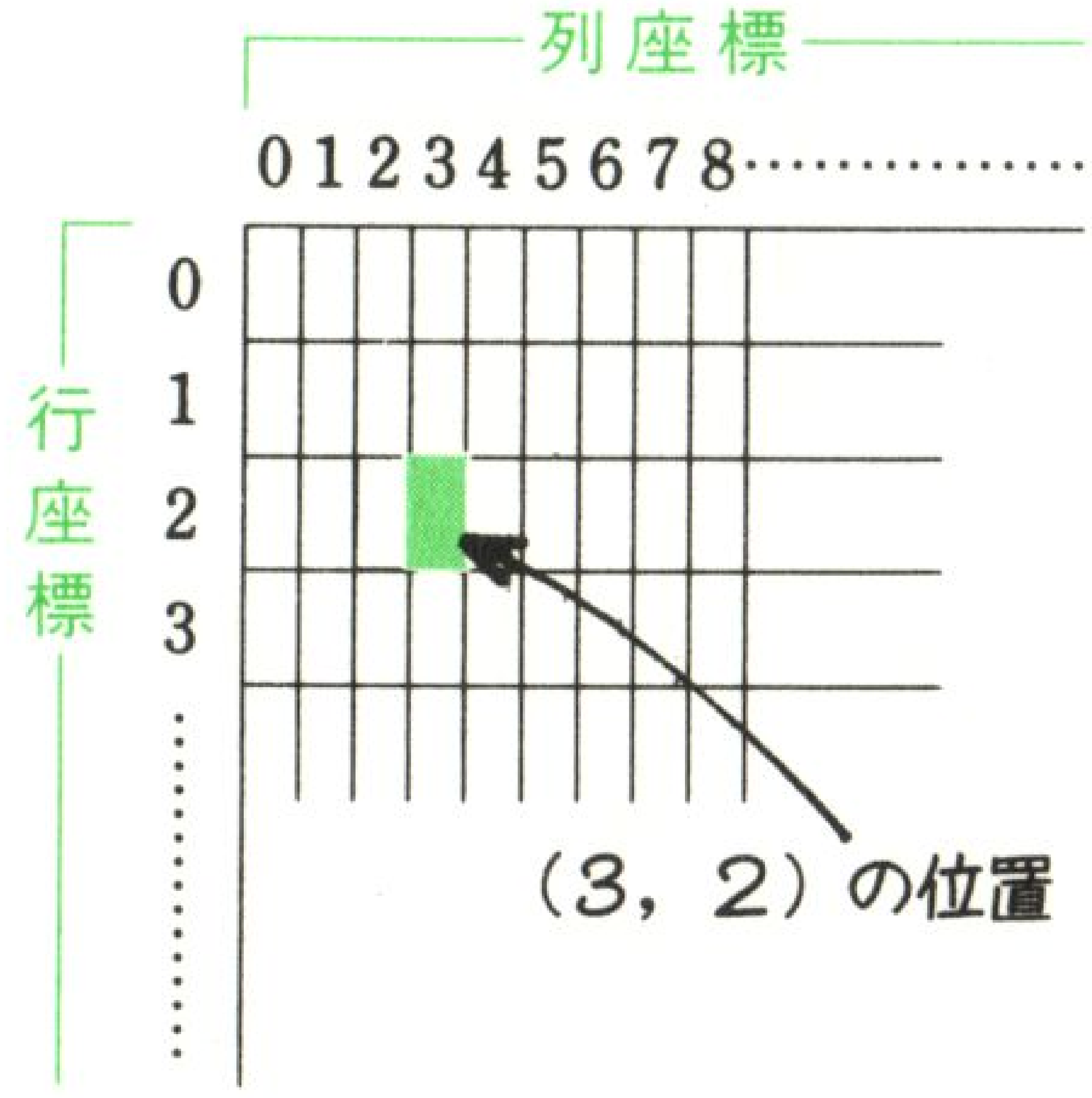
（X，Y）➡円を描く中心となる座標位置を指定

X＝列座標の番号を指定
Y＝行座標の番号を指定

〈半径〉➡描く円の半径をドットで指定

〈パレット番号〉➡描く円の色を指定

※パレット番号1から6までは132頁のカラーコードとおなじ



CIRCLE文と（3，2）の位置

■CIRCLE文は円を描く

グラフィック画面に円を描くには、CIRCLE（サークル）文を使います。CIRCLE文は、つぎのような形をしています。

CIRCLE(X,Y),〈半径〉,〈パレット番号〉

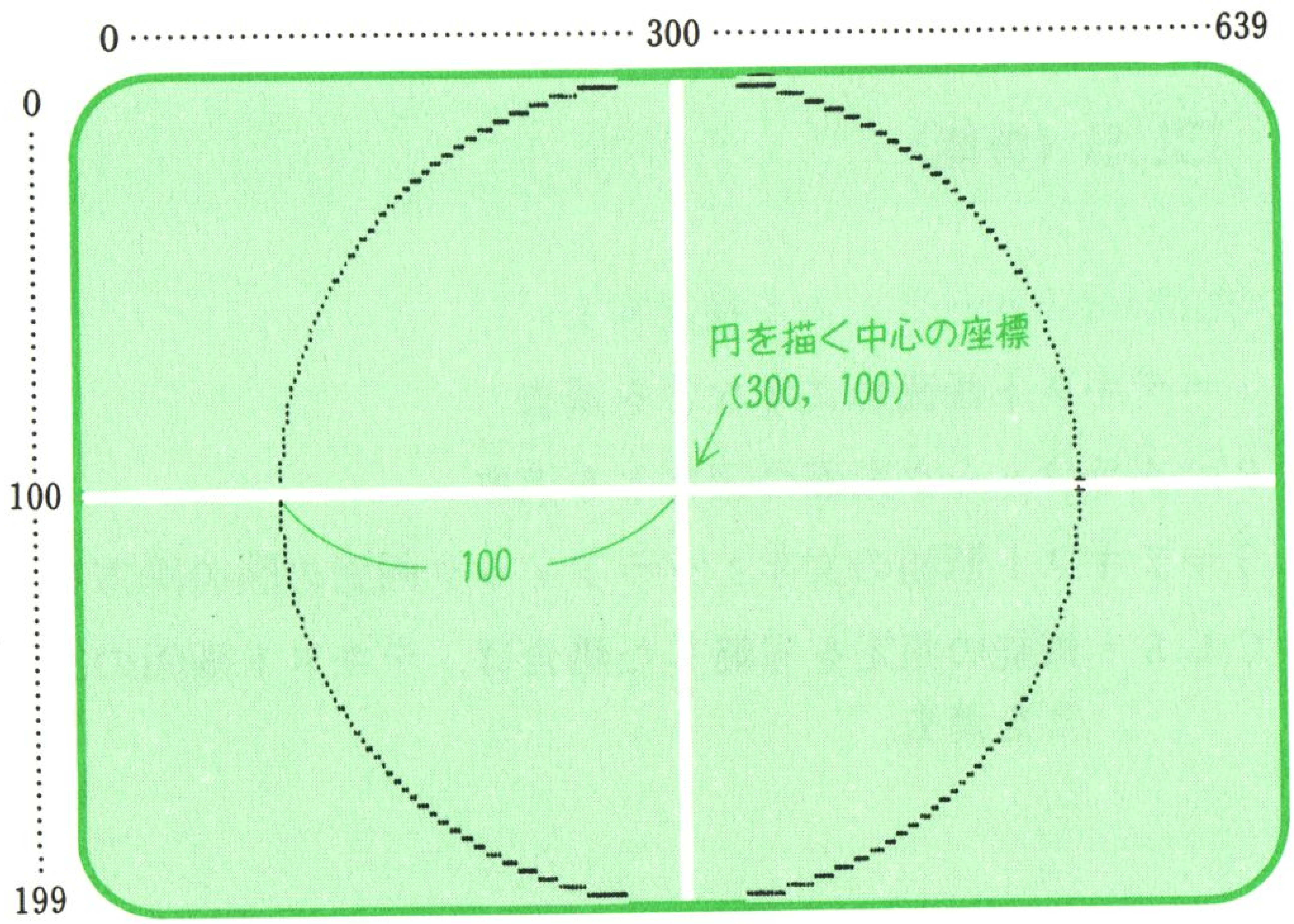
CIRCLE文のカッコのなかのXとYには、円を描く中心となる座標位置を指定します。座標位置の指定は、136頁に示した列座標と行座標の番号で指定します。Xには列座標の番号、Yには行座標の番号を指定します。

たとえば、Xを列座標の3、Yを行座標の2と指定すると、上の図に示す座標位置になります。

もちろん、この座標位置では画面の左上すぎて円を描くことができませんから、ここでは、中心の座標位置を列座標300、行座標100とします。したがって、Xのところに300、Yのところに100と指定します。

CIRCLE (300,100)←中心の座標位置を指定

〈半径〉のところには、描く円の半径をドット



CIRCLE (300, 100), 100, 3で描いた円

で指定します。〈パレット番号〉のところには、円を描く色をパレット番号で指定します。パレット番号1から6までは、COLOR文で示した1から6までのカラーコードとおなじですから、132頁を参照してください。

ここでは円を描く半径を100ドット、色を赤〈パレット番号2・カラーコードでも2〉とします。したがって〈半径〉のところに100、〈パレット番号〉のところに2を指定します。

CIRCLE (300, 100), 100, 2

これを入力して⏎キーを叩くと、SCREENで指定したグラフィック画面に、上に示すように列座標300、行座標100の座標位置を中心にして、半径100の円が赤で描かれます。

■CLS 3はグラフィック画面の図を消す

さて、グラフィック画面に描いた図などは、[HOME CLR]キーを押しても消すことができません。

試しに[HOME CLR]キーを押してみると、入力したSCREEN文やCIRCLE文は消え

CLS〈機能〉

〈機能〉➡1から3までを指定する。

1＝テキスト画面の文字などを消去

2＝グラフィック画面の図などを消去

3＝テキスト画面の文字とグラフィック画面の図の両方を消去

CLS＝機能の指定を省略した場合は、テキスト画面の文字などを消去

画面消去のCLS文

ますが、円は表示されたままです。

グラフィック画面に表示されている図を消去するには、CLS（クリアスクリーン）文を使います。CLS文は、つぎに示すような形をしています。

CLS〈機能〉

CLS文の〈機能〉のところに1から3まで指定できますから、2を入力し、⏎キーを叩くと、

CLS 2⏎ ←グラフィック画面の図を消去

グラフィック画面に表示されている図が消去されます。〈機能〉のところに3を指定すると、

CLS 3⏎ ←図と文字を消去

グラフィック画面に表示されている図と、テキスト画面に表示されている文字の両方が消去されます。〈機能〉のところに1を指定すると、

CLS 1⏎ ←テキスト画面の文字を消去

テキスト画面に表示されている文字が消去されます。この場合、CLSだけを入力して、

CLS⏎ ←テキスト画面の文字を消去

⏎キーを叩いてもおなじです。

第4章

BASICによるプログラミング



■ **プログラムモードはプログラムで動かすこと** ■

パソコンをプログラムで動かす

前の章でダイレクトモードについて示しましたが、電卓でできるような簡単な計算ならともかく、ふつうダイレクトモードで、パソコンに仕事を行わせるようなことはしません。プログラムモードで仕事を行わせます。

プログラムモード

プログラムモードとは、仕事の手順を書いたプログラムでパソコンに仕事を行わせることです。

プログラムなどというとむずかしく聞こえるかもしれませんが、ベーシックのプログラムは簡単で、ステートメントの先頭に行番号をつけるとプログラムになります。

●ステートメントの先頭に行番号をつけるとプログラムになる

```
10  SCREEN  0,0
20  CIRCLE  (300,100), 100, 2
```

●行番号は整数でつける

1　2　3　4　5　6　7　8　9　10　11　12　13

10　20　30　40　50　60　70　80　90　100　110　120

●小数点をともなった数字は、行番号にはつけられない

1.5　2.5　3.5　4.5　6.5　7.3　8.1　9.6

プログラムと行番号

たとえば、前の章のダイレクトモードで円を描きましたが、それをプログラムにするなら、上に示すように、ＳＣＲＥＥＮ文とＣＩＲＣＬＥ文の二つを実行させる順に並べて、さきに実行させるＳＣＲＥＥＮ文に10、つぎに実行させるＣＩＲＣＬＥ文に20というように、それぞれの先頭に行番号をつければよいのです。

行番号で実行する順番を示す

行番号は、整数でステートメントにつけることに決まっていて、小数点をともなった数字はつけることはできません。

したがって行番号は、１　２　３　４　５　６　７　８　９　10　11でもよいわけですが、ふつう　10　20　30　40　50　60　70　80　90　100　110というように、10おきにつけます。

このように行番号をつけると、パソコンは、入力されたステートメントをプログラムであると判断して、行番号の小さいほうのステートメントから順に記憶場所に記憶していきます。

またプログラムを実行する場合も、この行番号

```
10 SCREEN 0, 0       ←エラーにならない
10 SCREEN0, 0        ←エラー

10 CLS 3             ←エラーにならない
10 CLS3              ←エラー

10 WIDTH 40, 20      ←エラーにならない
10 WIDTH40, 20       ←エラー

20 CIRCLE(300, 100), 100, 2     }空白がなくてもエラーにならない
20 PRINT"ゲーム スタート"       }
```

空白がないとエラーになる命令と、ならない命令

の小さいほうのステートメントから順番に、メモリから取り出して実行していきます。

■⏎キーは行の終わりを知らせる

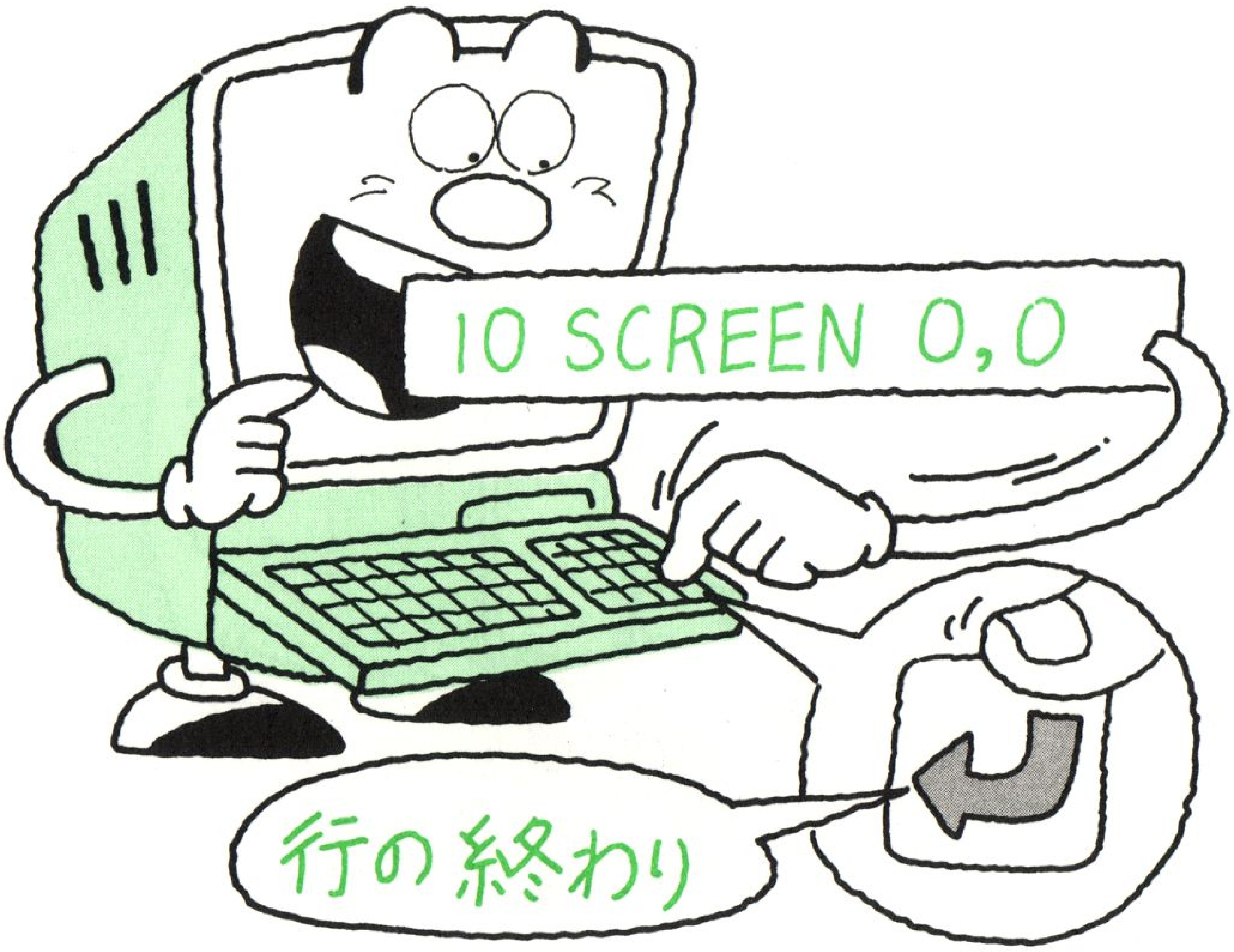
ステートメントに行番号をつけてプログラムにしたら、つぎはプログラムの入力の仕方です。

10 SCREEN 0, 0

行番号の「10」は、テンキーの1と0のキーを叩いて入力します。つぎの空白は、スペースキーを叩き、「SCREEN」は、アルファベットのキーを叩いて入力します。行番号のあとに空白を入力して一文字分あけるのは、プログラムを見やすくするためです。

SCREENのあとも、スペースキーを叩いて空白を入力してから、テンキーを叩いて「0、0」を入力します。この場合の空白は、プログラムを見やすくするためではなく、空白をあけないとエラーになるからです。

PC−9801などの場合、上に示すように、空白をあけないとエラーになる命令もあるので注意する必要があります。



ステートメントを入力し終わったら[⏎]キーを叩く

行番号10のステートメントを全部入力したら、[⏎]キーを叩きます。

10 SCREEN 0, 0 [⏎] ←必ず[⏎]キーを叩く

[⏎]キーが叩かれると、パソコンは、一行分のステートメントの入力が終わったことを知って、入力されたステートメントの最後に、行の終わりを示す00という記号をつけてメモリに記憶します。

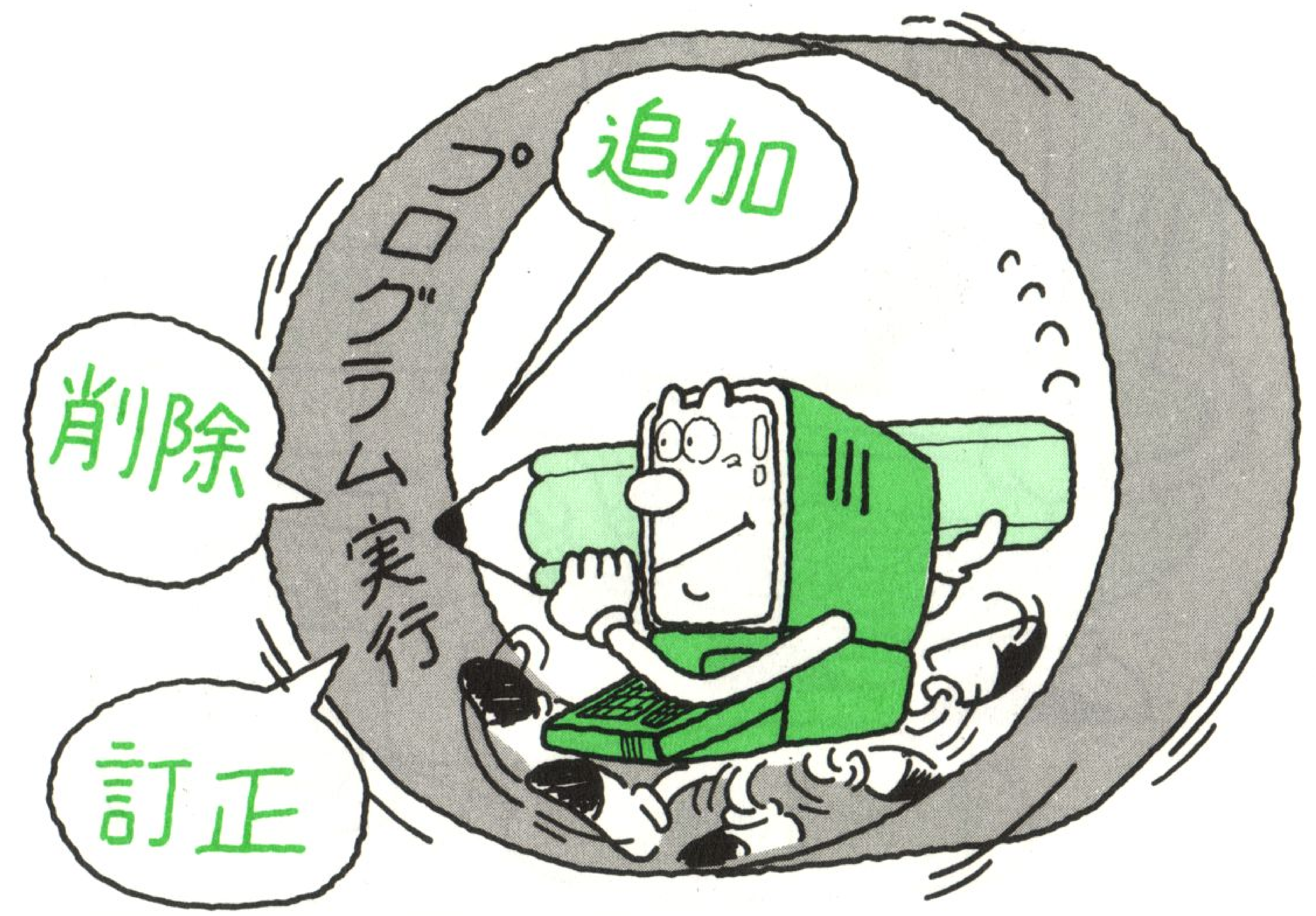
このように、一つのステートメントを入力し終わったら[⏎]キーを叩くということを繰り返して、プログラムを入力していきます。

プログラムの実行

プログラムの最後のステートメントを入力して[⏎]キーを叩いたら、つぎはプログラムの実行の方法です。

プログラムを実行させるコマンドはRUNコマンドですから、アルファベットの[R][U][N]とキーを叩いて[⏎]キーを叩くか、ファンクションキーの[f・5]キーを押します。

すると、メモリに記憶されているプログラムが実行されます。



ベーシックはステートメントを追加，訂正できる

プログラミングに必要なステートメントの追加・削除

ベーシックは、対話型のプログラミング言語ですから、完成されたプログラムを実行させるばかりではなく、未完成のプログラムを実行させながら、必要なステートメントを追加したり、不必要と思われるステートメントを削除したり、また、ステートメントのあちこちを訂正して、プログラムを完成させていきます。

プログラムを実行させながら完成させていく場合に必要になるのが、ステートメントの追加、削除、訂正の仕方です。

ここでは、さきに示した円を描くプログラムにいろいろ手を加えて発展させながら、ステートメントの追加、削除、訂正について示すことにします。

LINE（X1，Y1）−（X2，Y2），〈パレット番号〉、BまたはBF

（X1，Y1）➡線を描きはじめる座標位置を指定

X 1 ＝列座標の番号を指定：Y 1 ＝行座標の番号を指定

（X2，Y2）➡線を描き終わる座標位置を指定

X 2 ＝列座標の番号を指定：Y 2 ＝行座標の番号を指定

パレット番号➡線を描く色を指定

B➡（Bを指定すると）線で箱を描く

BF➡（BFを指定すると）線で箱を描いたあと中を塗りつぶす

線を描くLINE文

ステートメントの追加

つぎの円を描くプログラムを実行させると、単に画面の中央に赤い円を描くだけです。

```
10 SCREEN 0,0
20 CIRCLE (300,100),100,2
```

そこで最初は、この円のまわりを線で囲むことにします。

LINEは線を描く

グラフィック画面に線を描くには、LINE（ライン）文を使います。LINE文は、上に示すような形をしています。

LINE文の最初の（X1，Y1）には、線を描きはじめる座標位置を指定します。

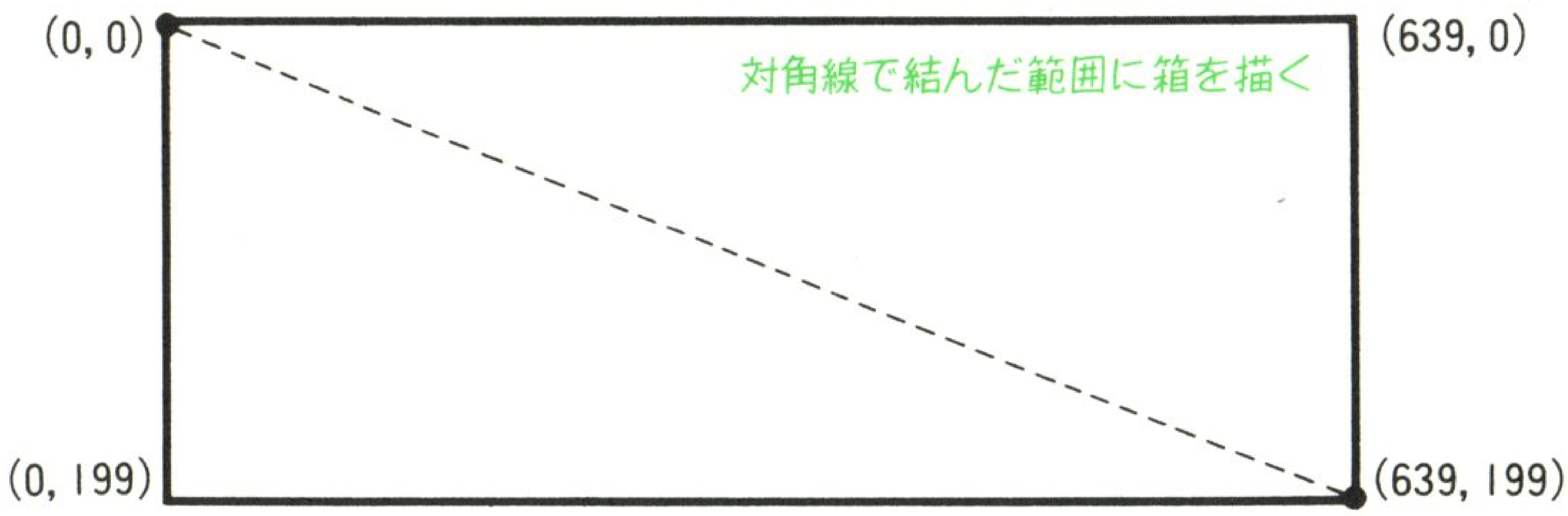
X1には列座標の番号、Y1には行座標の番号を指定します。

つぎの（X2，Y2）には、線を描き終わる座標位置を指定します。X2には列座標、Y2には行座標を指定します。

```
LINE (0, 0)－(639, 0), 1
```
← ヨコに線を描く




```
LINE (0, 0)－(639, 199), 1, B
```
← 枠（箱）を描く



LINE文で線や箱を描く（1）

〈パレット番号〉のところには、線を描く色をパレット番号で指定します。パレット番号の1から6までは、カラーコードとおなじです。

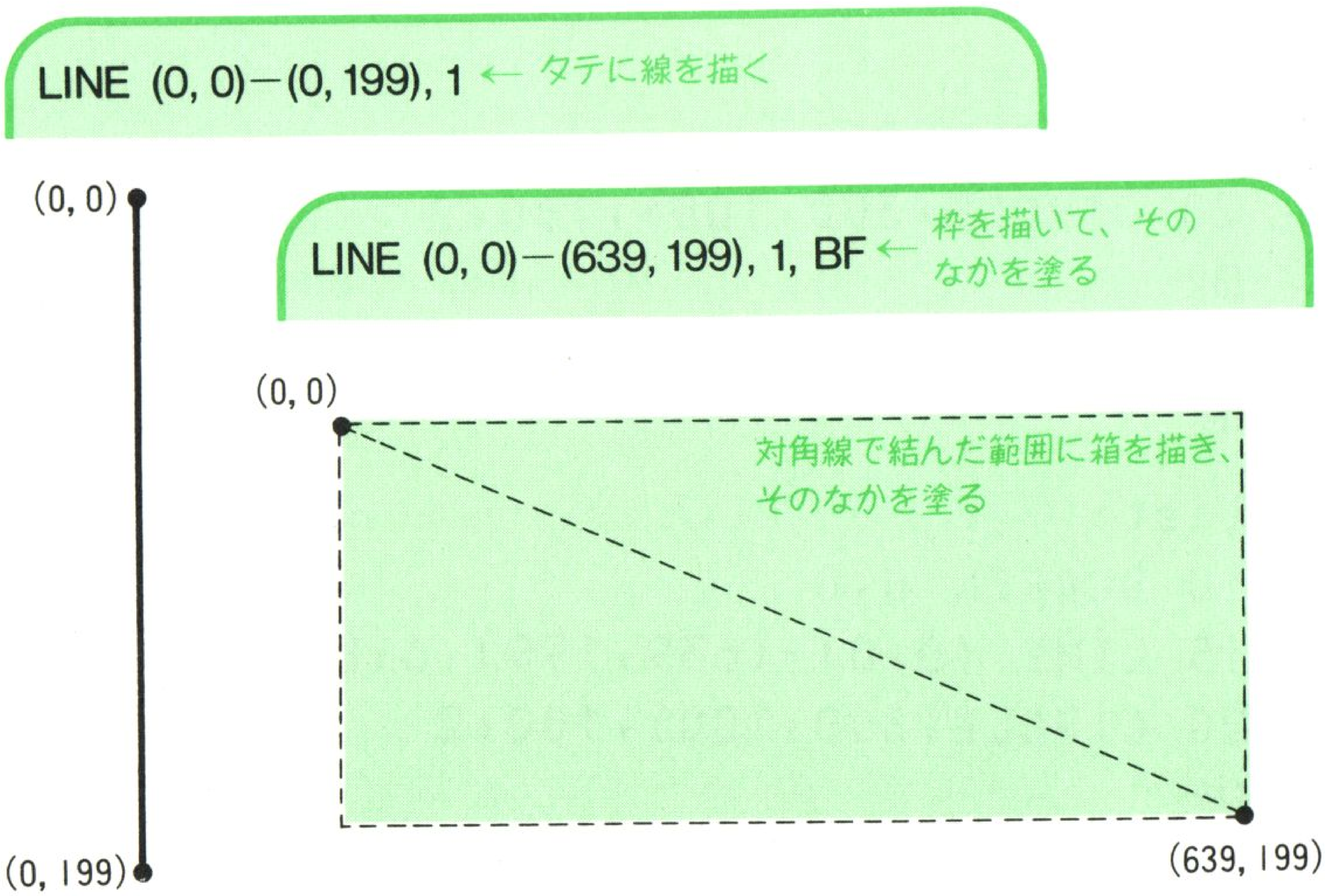
たとえば、画面の一番上、左端から右端まで青で線を描くとすれば、画面の上の左端の列座標は0、行座標も0、画面の上の右端の列座標は639、行座標は0ですから、最初のカッコのなかのX1に0、Y1に0を指定し、つぎのカッコのなかのX2に639、Y2に0を指定します。

青のパレット番号はカラーコードとおなじ1ですから、〈パレット番号〉のところに1を指定します。

```
LINE (0, 0) - (639, 0), 1
```

これで上に示すように、列座標0、行座標0の座標位置と列座標639、行座標0の座標位置を結んで青で線が描かれます。

LINE文の最後にBを指定すると、X1とY1に指定した座標位置と、X2とY2に指定した座標位置を対角線で結んだ範囲に、線で枠（箱）を描きます。



LINE文で線や箱を描く（2）

たとえば、画面一杯に枠を描くとすれば、左上角の列座標は０、行座標も０、画面の右下角の列座標は639、行座標は199ですから、左上角の列座標０をＸ１に、行座標０をＹ１に指定し、右下角の列座標639をＸ２に、行座標199をＹ２に指定します。

ＬＩＮＥ（０，０）－（639，199），１，Ｂ

そして〈パレット番号〉のところを１、最後にＢと指定すると、前頁に示すように、（０，０）の座標位置と（639，199）の座標位置を対角線で結んだ範囲に青で枠を描きます。

ＬＩＮＥ文の最後の指定を、ＢＦと指定した場合は、上に示すように、まずＢと指定したときとおなじように枠を描き、つぎに枠を描いた色で枠のなかが塗りつぶされます。

ここでは円を描くプログラムに、最後にＢＦと指定したＬＩＮＥ文を追加することにします。

描く枠の大きさは画面一杯とします。枠を描く色は、枠のなかを塗りつぶす関係から、黄色とします。したがって、追加するＬＩＮＥ文は、つぎのようになります。

```
list⏎
10 SCREEN 0,0
20 CIRCLE(300,100),100,2
Ok
15 line (0,0)-(639,199),6,bf ⏎
■

list⏎
10 SCREEN 0,0
15 LINE (0,0)-(639,199),6,BF
20 CIRCLE(300,100),100,2
Ok
■
```

LINE文に15とつけて入力し、⏎キーを叩く

LISTコマンドでメモリのプログラムを表示

追加されたLINE文

ステートメントの追加

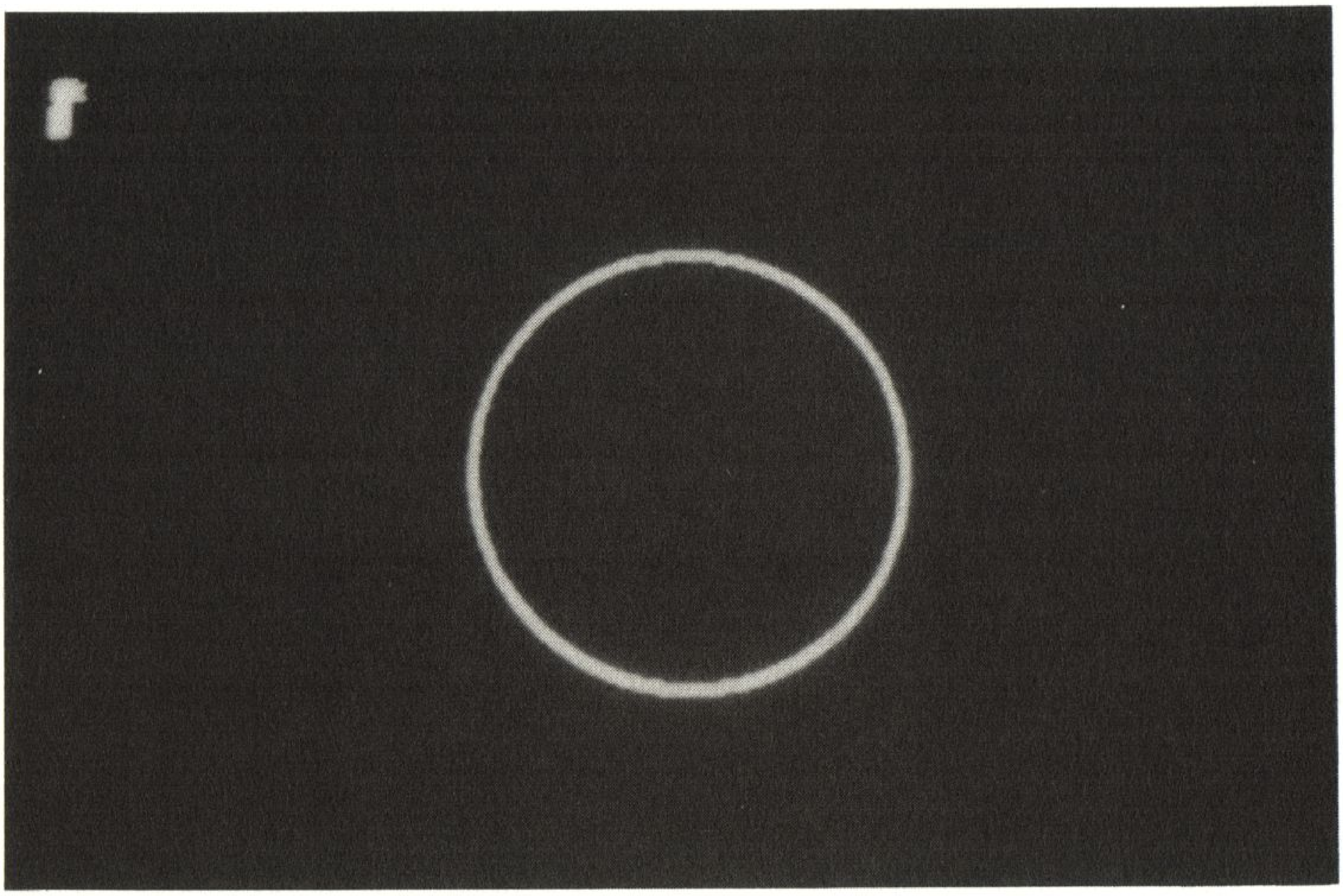
LINE (0, 0) − (639, 199), 6, BF

円を描くプログラムに、このLINE文を追加するには、行番号をつけなければなりません。ステートメントにつける行番号は、プログラムのどこに追加するかで決まります。

ここではLINE文を、行番号10のSCREEN文と、行番号20のCIRCLE文の間に追加することにします。したがって、LINE文につける行番号は、行番号10と行番号20の間の数字、つまり11から19までとなります。

この場合は中間を取って、行番号を15としますから、上に示すように、LINE文に15とつけて入力します。追加するステートメントの入力は、行番号20のLINE文の次に表示されているカーソルの位置で行います。入力し終わったら、⏎キーを叩きます。⏎キーを叩き忘れると、追加されません。

これでLINE文は、上に示すように、行番号10と20の間に追加されます。行番号を10おきにつけておくと、このように間の行番号に必要なステ



プログラムの実行結果

ートメントが追加できるので便利です。

　LINE文を追加したプログラムをRUNコマンドで実行させると、入力した文字やファンクションキーの内容が表示されたままで、黄色地に赤で円が描かれるので、写真に示すようにすっきりとは描かれません。

　このように描くには、画面をクリアする命令CLS（クリアスクリーン）に3を指定して行番号3とし、ファンクションキーの内容の表示を消去する命令CONSOLE（コンソール）に0を指定して行番号6として、ステートメントを追加します。

```
3 CLS 3
6 CONSOLE,,0
```

RENUMで行番号をつけなおす

　ステートメントを追加すると、プログラムの行番号は3、6、10、15、20と不揃いになります。

　プログラムの行番号がどのようになっていても、行番号の小さいステートメントから順に実行されるので問題ありませんが、このように行番号がつ

```
renum ↵        ← RENUMコマンドを入力して↵キーを叩く
Ok             ← 行番号をつけなおしたOK
list ↵         ← メモリのプログラムを表示させる
10 CLS 3
20 CONSOLE ,,0
30 SCREEN 0,0
40 LINE (0,0)-(639,199),6,BF
50 CIRCLE (300,100),100,2
Ok
■
```

（10〜50の行：行番号がつけなおされたプログラム）

RENUMコマンドで行番号をつけなおす

いていると、ステートメントを追加・削除・訂正する場合に、間違いをおこしやすくなります。

そこで行番号を、10、20、30、40というように10おきにつけなおします。行番号をつけなおすのは、RENUM（リナンバー）コマンドです。

RENUMコマンドは、[R][E][N][U][M]と入力して[↵]キーを叩くだけです。これで行番号は、上に示すように、10から10おきにつけなおされます。

このように、RENUMコマンドでときどき行番号をつけなおしながら、ステートメントを追加していくわけです。

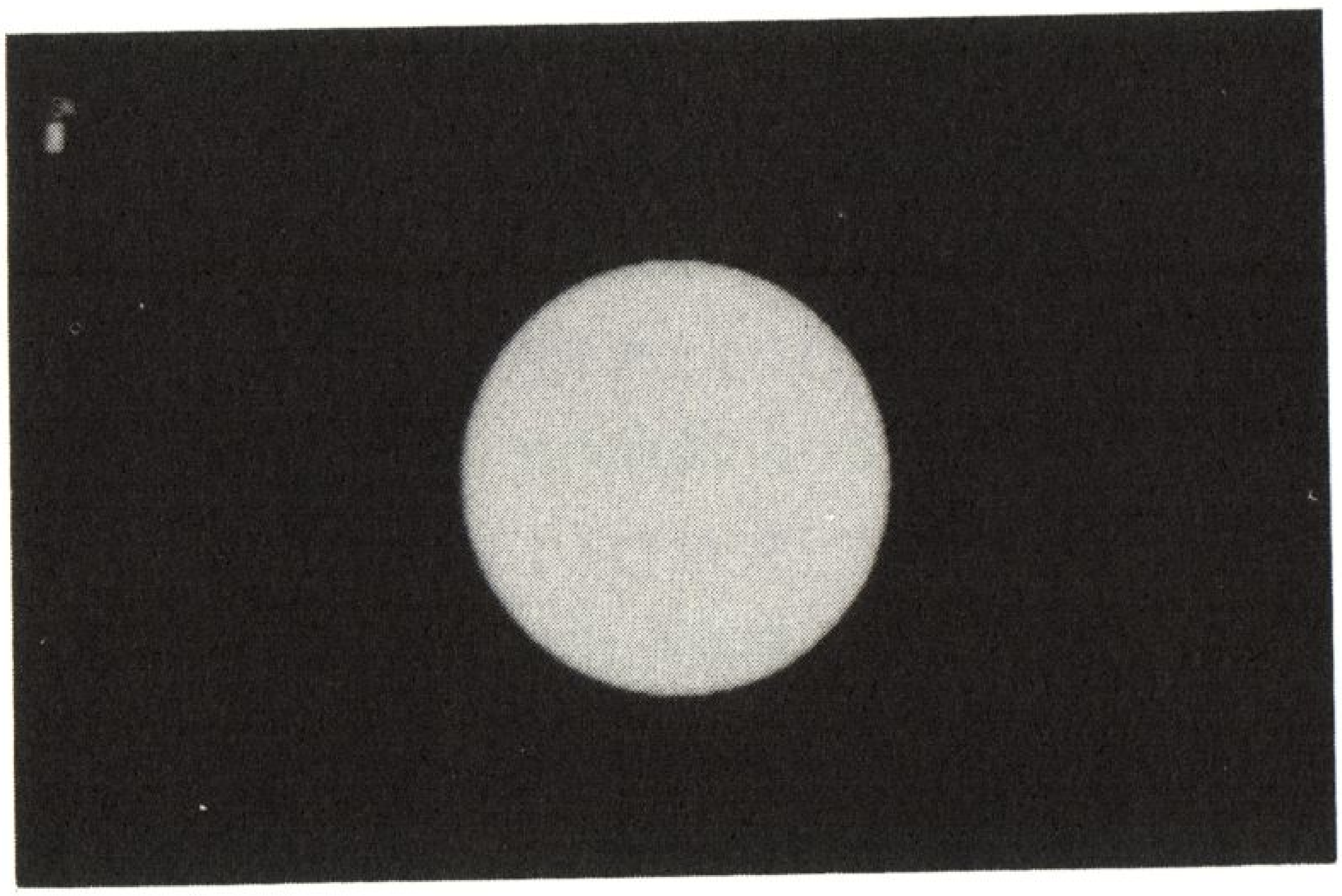
RNDは乱数を作る

つぎは、円を描く地の色を黄色だけでなく、いろいろな色に変えます。また、描いた円のなかも色で塗り、その色もいろいろに変えていくことにしますから、さらにステートメントを追加していきます。

まず、描いた円のなかを塗ります。

円のなかを塗るには、CIRCLE文の〈パレット番号〉のあとに、4個の「,」コンマを入力

CIRCLE（X，Y），〈半径〉，〈パレット番号〉,,,,F



4個の「，」コンマを入力してFと指定すると円のなかが塗られる

CIRCLE文にFを指定

してFと指定します。この場合は、つぎのようになります。

50　CIRCLE (300, 100), 100, 2,,,,F

4つの「，」コンマとFを入力するには、カーソルキーでカーソルを行番号50のCIRCLE文の最後の2のあとに移動させてテンキーの[,]キーと[F]キーを叩けばよいのです。「，」とFを入力したら[↵]キーを叩いて、改めて行番号50のCIRCLE文を入力します。これで、赤で円を描いたあと、円のなかが赤で塗られます。

では地の色と円の色を、いろいろな色に変えます。色を変えるには、乱数を使います。

乱数を作るのは、RND（ランダム）関数です。RND関数は関数ですから単独では使わずに、RND関数と変数を＝で結んで使います。

A＝RND（1）

このようにすると、RND（1）で作り出された乱数が変数Aに代入されます。

RND（1）で作り出される乱数は0.000000から0.999999ですから、そのままでは乱数として使

〈変数〉＝ RND (X) ←— 変数とRNDを＝で結んで使う

● X＝1を指定すると乱数を作る

① A＝RND(1)

0.000000から0.999999までの乱数が作り出される

② A＝(RND(1)＊10)

10倍すると、0.000000から9.99999までの乱数が作り出される

③ A＝(RND(1)＊10)＋1

＋1すると、1.000000から10.99999までの乱数が作り出される

④ A＝INT(RND(1)＊10)＋1

INTで小数点を切り捨てて1から10までの乱数となる

⑤ A＝INT(RND(1)＊6)＋1

＊10を＊6にすると、1から6までの乱数となる

RND関数

えません。そこで、1から10までの乱数を作るときは、上の③に示すように10倍し、それに＋1します。

この場合は1から6までの乱数を作るので、⑤に示すように6倍し、それに＋1します。

A＝(RND(1)＊6)＋1

これで1.000000から6.99999までの乱数が作られますから、INT（インテジャー）関数で小数点以下を切り捨てて、1から6までの整数の乱数にします。

A＝INT(RND(1)＊6)＋1

1から6までの乱数は、変数Aに代入されます。

ところで、地の色と円を描く色を変えるためにもう一つ1から6までの乱数を作ります。作られた1から6の乱数は、変数Bに代入します。

B＝INT(RND(1)＊6)＋1

このふたつの乱数を作るステートメントをプログラムに追加します。

変数Aに代入した1から6までの乱数は、行番号40のLINE文の〈パレット番号〉に与えて地

```
list
10 CLS 3
20 CONSOLE ,,0
30 SCREEN 0,0
40 LINE (0,0)-(639,199),A,BF
50 CIRCLE (300,100),100,B,,,,F
Ok
34 a=int(rnd(1)*6)+1 ⏎
38 b=int(rnd(1)*6)+1 ⏎
■
```

カーソルを6の上にもってきてAに訂正

カーソルを2の上にもってきてBに訂正

二つの乱数をつくるステートメントを追加

乱数のステートメントの追加とステートメントの訂正

の色を指定し、変数Bに代入した1から6までの乱数は、行番号50のCIRCLE文の〈パレット番号〉に与えて、円を描く色を指定します。

したがって、乱数を作る二つのステートメントは、行番号30のSCREEN文と行番号40のLINE文の間に入れますから、上に示すように行番号34、行番号38として追加します。

追加し終わったら、行番号40のLINE文の、〈パレット番号〉に指定されている6と、行番号50のCIRCLE文の〈パレット番号〉に指定されている2を、AとBに訂正します。

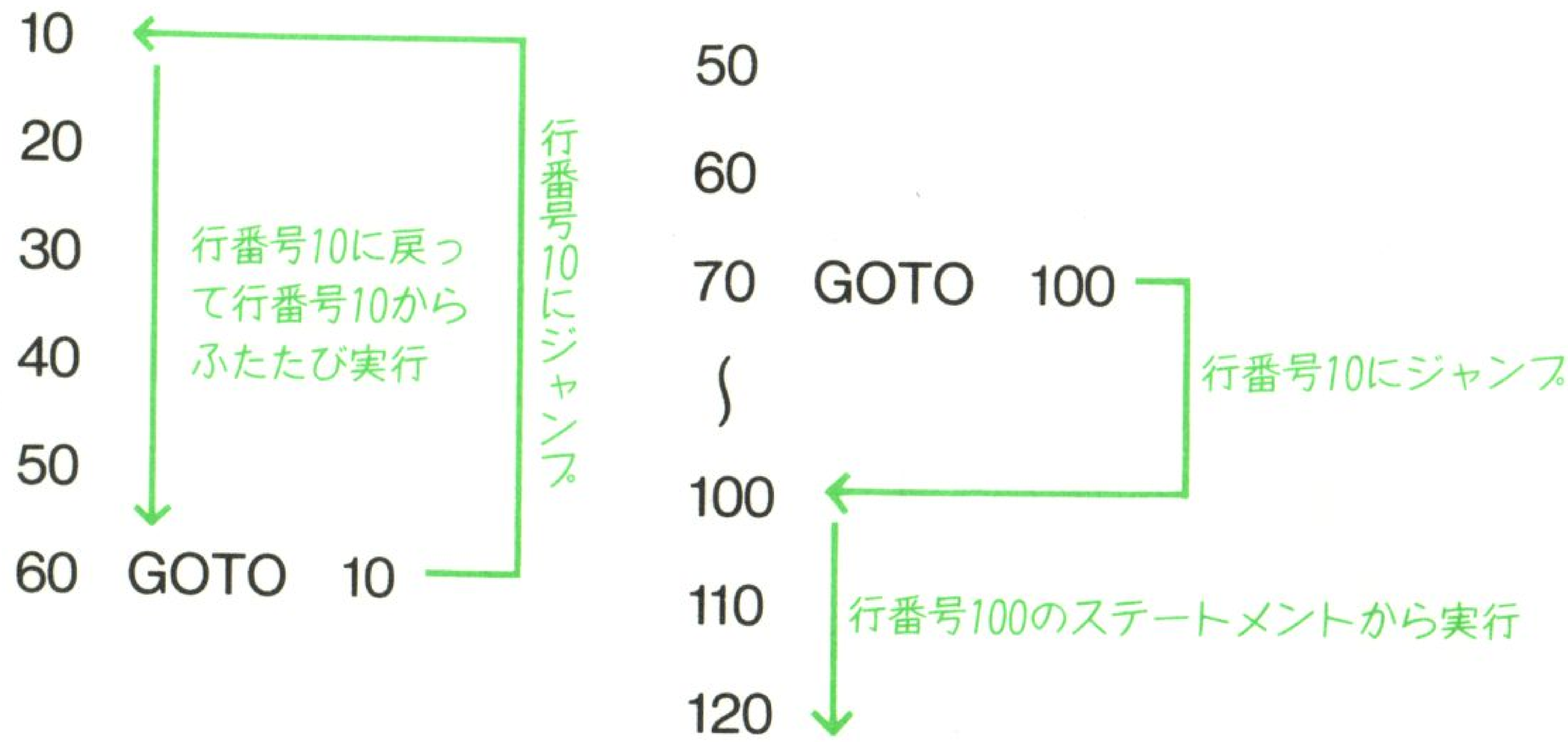
訂正の仕方は、カーソルキーでカーソルを行番号40の6の上に移動して[A]キーを叩き、Aの文字が入力されたら[⏎]キーを叩きます。行番号50の2をBに訂正する場合もおなじです。

訂正が終わったら、カーソルキーでカーソルをプログラムの下に移動して、[f・5]キーを押してRUNコマンドを入力します。

プログラムの上でRUNコマンドを入力すると、エラーになります。

GOTO〈行番号〉

〈行番号〉➡　ジャンプする行番号を指定



■GOTO文の働き

RUNコマンドを入力すると、行番号34と38の乱数を作るステートメントで、そのとき作り出された乱数が変数AとBに代入されて、行番号40のLINE文の変数Aと行番号50のCIRCLE文の変数Bに送られて色を指定します。

たとえば、行番号34で変数Aに乱数の1、行番号38で変数Bに乱数の5が代入されたとすると、1が行番号40のLINE文の変数Aに、5が行番号50のCIRCLE文の変数Bに送られて色を指定しますから、青の地に水色の円が描かれます。

ところでこのプログラムを、繰り返しRUNコマンドを入力して実行させても、最初に描かれた色で描かれて、いっこうに色は変化しません。そこでGOTO文で繰り返すことにします。

■GOTOは指定した行番号に飛ぶ

GOTO文は、上に示すような形をしていてGOTO文に指定した行番号に無条件にジャンプします。

そして、ジャンプした先の行番号のステートメントから実行していきます。

```
list⏎
10 CLS 3
20 CONSOLE ,,0
30 SCREEN 0,0
34 A=INT(RND(1)*6)+1
38 B=INT(RND(1)*6)+1
40 LINE (0,0)-(639,199),A,BF
50 CIRCLE (300,100),100,B,,,,F
Ok
60 for t=0 to 3000:next⏎ } ウェイトループと
70 goto 34⏎              } GOTO文を追加
```

GOTO文とウエイト・ループを追加して完成したプログラム

ＧＯＴＯ文の〈行番号〉に指定する行番号は、まえのほうの行番号であってもあとのほうの行番号であってもかまいません。

このプログラムでは、行番号34と38の乱数を作るステートメントと、行番号40と50のＬＩＮＥ文とＣＩＲＣＬＥ文を繰り返すと、地の色がいろいろな色に変化し、そのうえにいろいろな色で円が描かれるわけですから、ＧＯＴＯ文でジャンプさせる行番号は34となります。

GOTO 34

このＧＯＴＯ文を追加するところは、行番号50のＣＩＲＣＬＥ文を実行したあと行番号34にジャンプさせるので、ＣＩＲＣＬＥ文のつぎに追加すればよいのですが、行番号60にＧＯＴＯ文を置くと、非常な速さでステートメントが実行されてしまって、色の変化がわかりません。

そこで、ウエイト・ループ（164頁参照）を行番号60として追加し、行番号70としてＧＯＴＯ文を追加します。

これでプログラムは完成です。

```
34⏎
40⏎
■
list⏎
10 CLS 3
20 CONSOLE ,,0
30 SCREEN 0,0
38 B=INT(RND(1)*6)+1
50 CIRCLE (300,100),100,B,,,,F
60 FOR T=0 TO 3000:NEXT
70 GOTO 34
Ok
```

（34⏎・40⏎）行番号を入力して⏎キーを叩くとステートメントが削除される

（10〜70）行番号34と40を削除した結果

ステートメントの削除

ステートメントの削除

これまで、ステートメントの追加の仕方を示しながらプログラムを作ってきましたが、つぎは、不必要なステートメントの削除の仕方です。

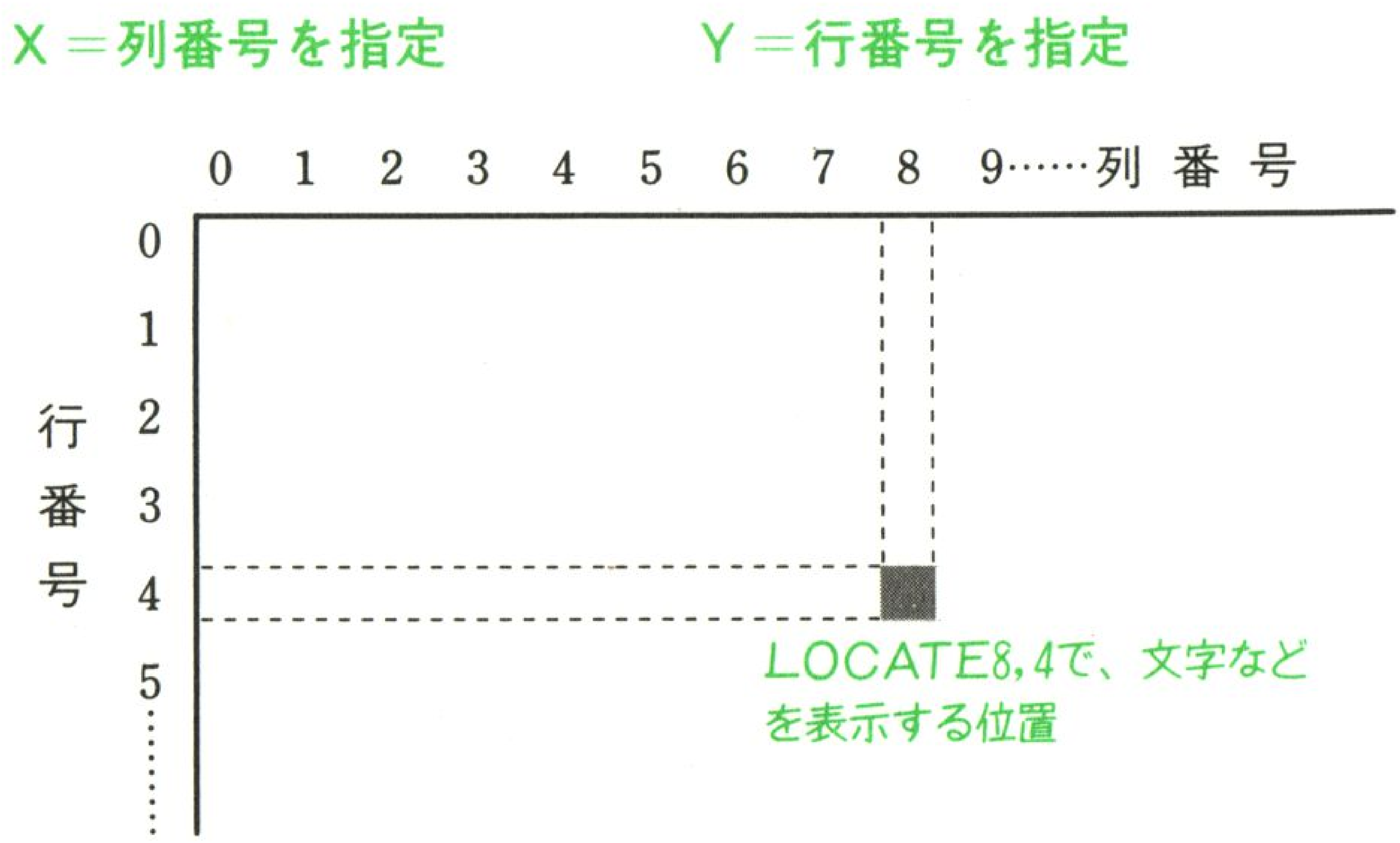
不必要なステートメントを削除するには、削除する行番号を入力して、⏎キーを叩けばよいのです。

たとえば、まえの頁に示したプログラムのステートメントのうち、行番号34の乱数を作るステートメントと、行番号40のＬＩＮＥ文を削除するとするなら、上に示すように、34を入力して⏎キーを叩きます。

これで、行番号34の乱数をつくるステートメントが削除されましたから、つぎに40と入力して⏎キーを叩きます。これで行番号40のＬＩＮＥ文が削除されます。

ＬＩＳＴコマンドでメモリのプログラムを表示させた結果を示します。

LOCATE　X，Y



LOCATE文

プログラミングで使うステートメント

さきの項で、プログラムを作るのに必要なステートメントの追加と削除の仕方について示しました。ここでは、プログラムを作るうえで必要なステートメントと、その働きについて示します。

LOCATEは表示位置を指定する

表示させる文字などを、画面の中央などに表示させようとするときは、PRINT文で表示させる文字などの位置を指定しなければなりません。

このときに使うのが、LOCATE（ロケート）文です。

LOCATE文は、上に示すような形をしています。

```
10 LOCATE 30,10
20 PRINT "ｹｲｻﾝ ﾉ ﾌﾟﾛｸﾞﾗﾑ"
```

実行結果

```
run



                                  ｹｲｻﾝ ﾉ ﾌﾟﾛｸﾞﾗﾑ
Ok
```

表示位置を指定して文字を表示

LOCATE文のXには、画面の横方向につけられた列番号、Yには、画面の縦方向につけられた行番号を指定します。するとカーソルは、Xに指定した列番号と、Yに指定した行番号が交わった位置に移動して、その位置からPRINT文で表示する文字などを表示します。

たとえば、「ケイサン ノ プログラム」の文字を画面のほぼ中央に表示させるとするなら、LOCATE文のXには列番号30を指定し、Yには行番号10と指定します。そして、文字を表示させるPRINT文をつぎにおきます。

10 LOCATE 30, 10
20 PRINT "ケイサン ノ プログラム"

これを入力して実行させると、うえに示すように、列番号30と行番号10が交わった位置にカーソルが移動して、その位置に「ケ」を表示し、つづけて「イサン ノ プログラム」を表示します。

LOCATE文で表示位置を指定するのは、つぎにおかれた一つのPRINT文だけです。なお、LOCATE文のXとYには、10とか20とかいう

FOR 〈変数〉=〈初期値〉 TO 〈最終値〉
～
NEXT 〈変数〉

変数＝数値変数を指定
初期値＝繰り返す最初の値を指定
最終値＝繰り返しを終える値を指定

```
10  FOR  I=1  TO  10
20
30
40
50  NEXT  I
```

行番号20から40までのステートメントを、初期値1から最終値10までの10回繰り返し働かせる

FOR～NEXT文

定数だけでなく、変数を指定することができます。XとYに変数を指定する例は、つぎのFOR～NEXT文で示します。

FOR～NEXTは繰り返し処理

FOR～NEXT（フォア～ネクスト）文は上に示すようにFOR文とNEXT文にわかれていて、その間におかれたステートメントを繰り返し働かせます。

繰り返す回数は、FOR文の〈初期値〉と〈最終値〉に指定します。〈初期値〉には繰り返しをはじめる最初の値、〈最終値〉には、繰り返しを終える値です。

FOR文の〈変数〉には、AとかIなどの数値変数を指定します。NEXT文の〈変数〉は、指定を省略することができます。NEXT文の〈変数〉を指定するときは、FOR文の〈変数〉に指定した変数と、おなじ変数を指定します。違う変数を指定するとエラーになります。

```
10 FOR I=1 TO 25
20 PRINT "♥";
30 NEXT
```

実行結果

```
run
♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥♥
Ok
```

PRINT文を25回繰り返して♥を25個表示

FOR~NEXT文でPRINT文を繰り返して♥を表示

たとえば、画面に♥を横に25個並べて表示するとしたら、つぎのように、PRINT文に♥を1個の「〃」引用符で囲んでおきます。

PRINT "♥"

PRINT文の最後の「;」セミコロンは、文字などを横に並べて表示させる指定です。「;」をつけないと縦に表示されるので注意してください。そしてFOR~NEXT文で、このPRINT文を25回繰り返させます。

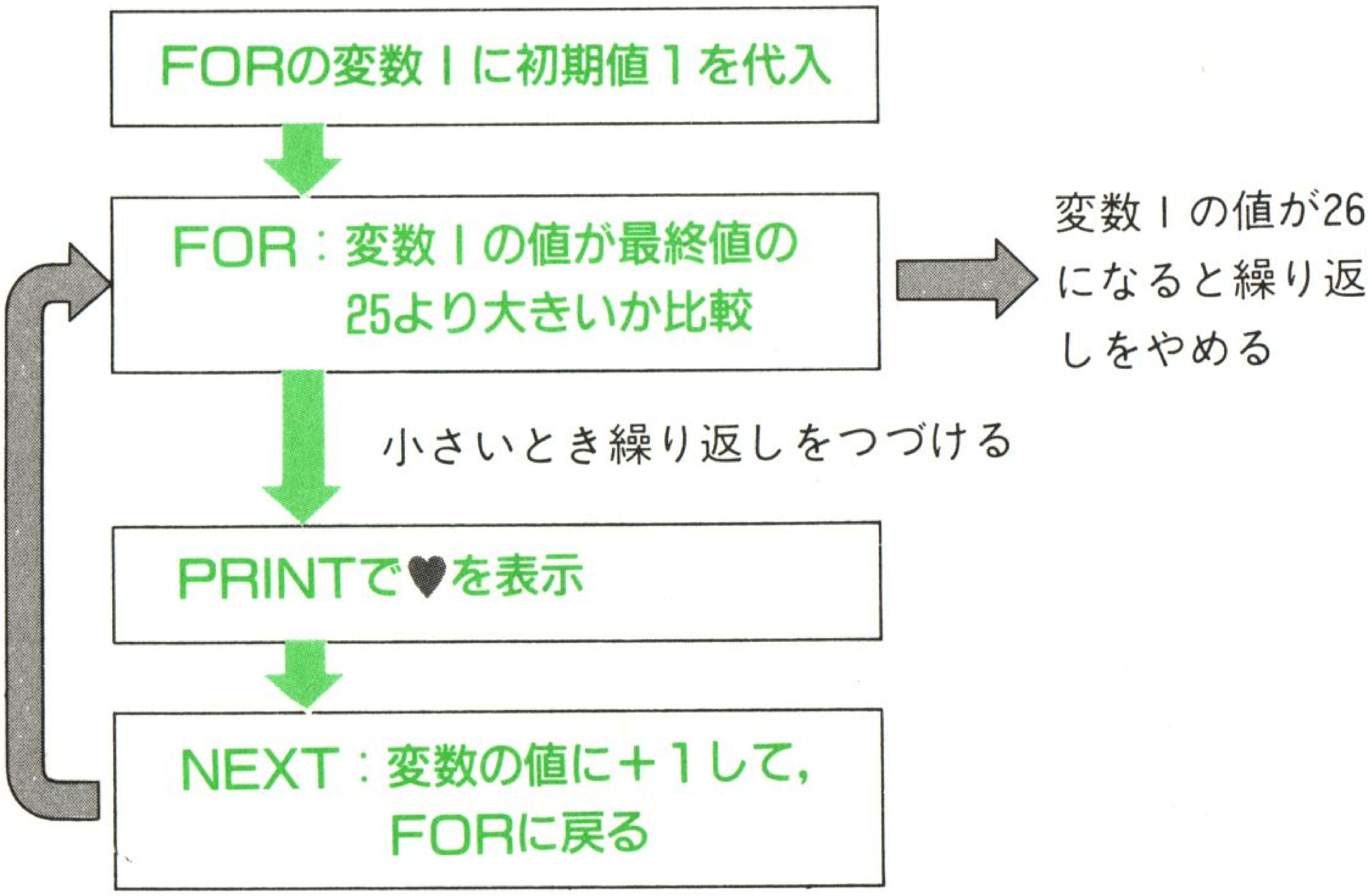
25回繰り返させるためには、FOR文の初期値を1、最終値を25と指定します。初期値を0、最終値を24と指定してもおなじです。

この場合、FOR文の変数をIとします。NEXT文の変数の指定は省略します。

```
FOR I=1 TO 25
 ∫
NEXT
```

このFOR~NEXT文の間に、上に示すように、PRINT文を入れます。

これを実行させると、まず、FOR文の変数I



FOR～NEXTの繰り返し方

に初期値の１が代入されて、変数Ｉは１になります。つぎにＰＲＩＮＴ文が実行されて、１個目の♥を表示します。

　つぎにＮＥＸＴ文が実行されます。ＮＥＸＴ文は、ＦＯＲ文の変数の値に＋１する働きをします。

　この場合、変数Ｉの値は１ですから、１＋１を行って、その結果の２を変数Ｉの値にします。そして、ＦＯＲ文に戻ります。

　ＦＯＲ文は、変数Ｉの値が最終値よりも大きくなったかどうか、比較する働きがあります。

　この場合、変数Ｉの値は２になって戻ってきていますから、最終値25より大きくありません。したがってＰＲＩＮＴ文が実行されて、２個目の♥が表示されます。

　２個目の♥を表示すると、ＮＥＸＴ文を実行して、変数Ｉの値に＋１します。変数Ｉの値は２ですから、２＋１を行って、その結果の３を変数Ｉの値にします。

　そしてＦＯＲ文に戻って、変数Ｉの値が最終値25より大きいかどうか比較します。

FOR〈変数〉＝〈初期値〉TO〈最終値〉：NEXT

■FOR文とNEXT文の間を指定した回数行き来して、つぎのステートメントの実行を遅らせる

```
10 FOR I=1 TO 10
20 LOCATE 30,10
30 PRINT "oooooooooo"
40 FOR T=0 TO 3000:NEXT
50 LOCATE 30,10
60 PRINT "xxxxxxxxxxx"
70 FOR T=0 TO 3000:NEXT
80 NEXT
```

20：○の表示位置を指定
30：○を表示
40：ウエイト・ループ
50：×の表示位置を指定
60：×を表示
70：ウエイト・ループ
10～80：10回繰り返して○と×を交互に表示させる

ウエイト・ループ

このことを繰り返して、変数Iの値が最終値の25になって、PRINT文で25個目の♥を表示すると、つぎにNEXT文を実行します。

NEXT文が実行されると、変数Iの値は25ですから、25＋1で変数Iの値を26にします。

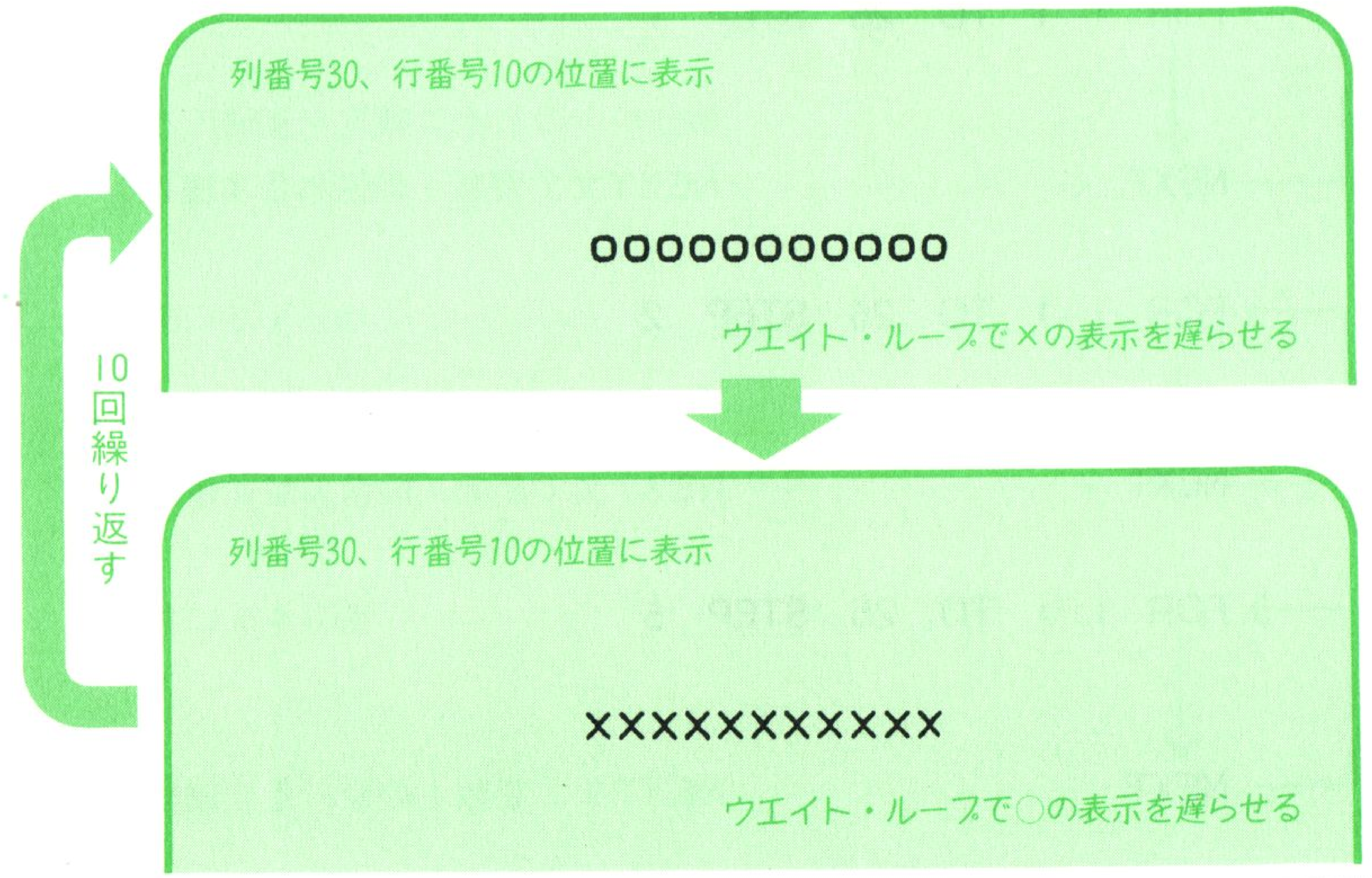
そしてFOR文に戻って、変数Iの値が最終値の25より大きいか比較します。こんどは、変数Iの値は26で最終値は25ですから、変数Iの値は最終値より大きいので、そこで繰り返しをやめて、NEXT文のつぎにある行番号のステートメントを実行します。

この場合は、NEXT文のつぎにステートメントがありませんから、プログラムの実行を終了し、OKが表示されます。

■ウエイト・ループ

FOR～NEXT文の働きを利用したものに、ウエイト・ループがあります。

ウエイト・ループは、上に示すような形をしていて、初期値に指定した値から、最終値に示した値まで、FOF文と「：」コロンでつながれたN



ウエイト・ループで○と×の表示する時間を調節する

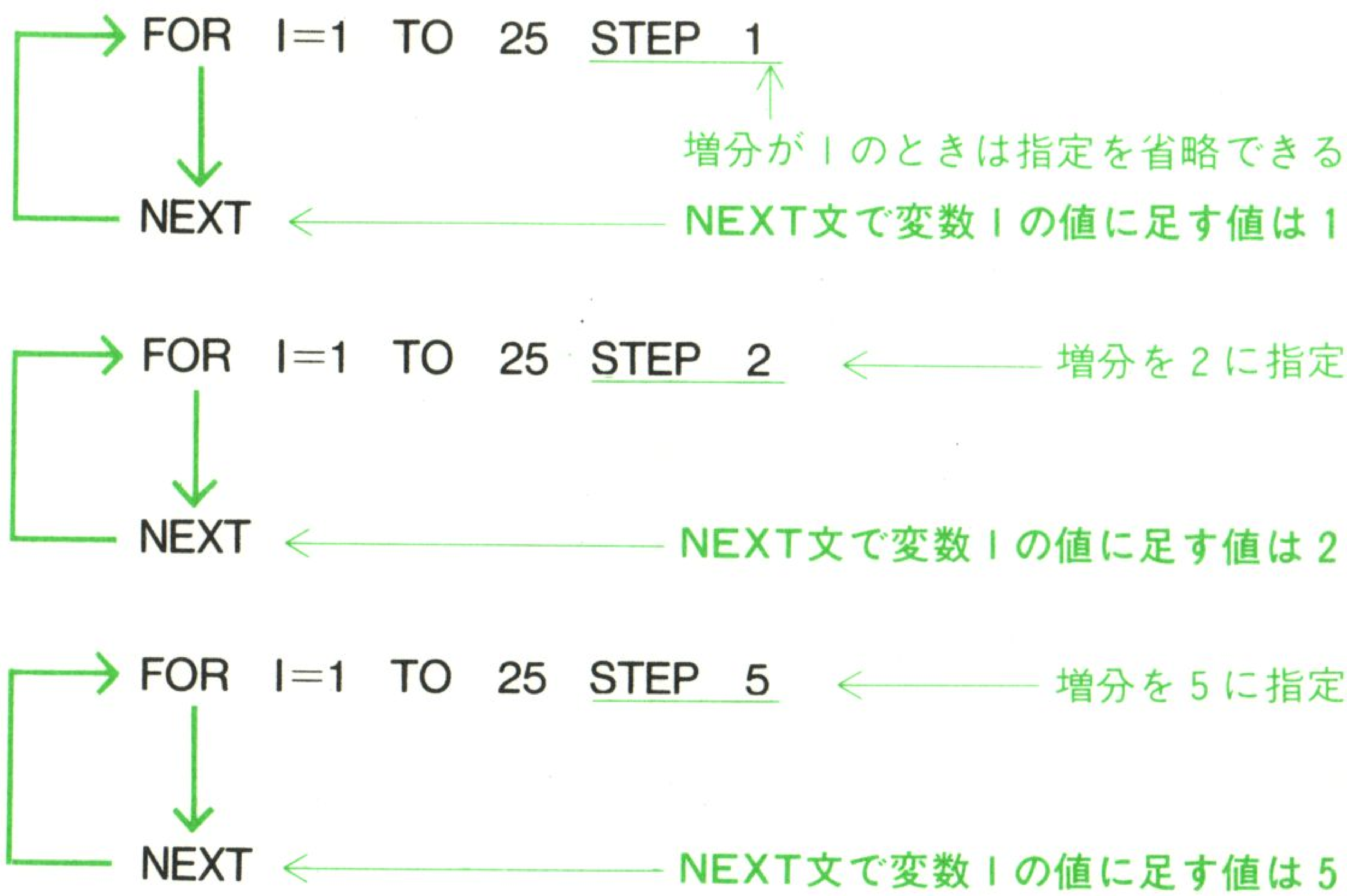
EXT文の間を、ただ行ったり来たりします。

したがって、ステートメントの実行がはやすぎて、なにが表示されているのかわからない場合、このウエイト・ループをステートメントの間に入れると、つぎのステートメントの実行が遅れるので、なにが表示されているかわかるようになります。

ステートメントの実行を、どのくらい遅らせるかはプログラムによって異なりますから、最終値を小さくしたり大きくしたりして調整します。

まえの頁に示したプログラムは、行番号10のFOR文と行番号80のNEXT文で、11個の○と×を交互に10回、繰り返して表示させるプログラムですが、この場合、行番号20と50のLOCATE文でおなじ位置を指定して、PRINT文で11個の○と×を表示しますから、ウエイト・ループがないと速く実行されて、○が表示されているのか×が表示されているのかわかりません。

そこで、行番号40と行番号70にウエイト・ループを入れて、○と×が交互に表示されている様子がわかるようにしています。



FOR～NEXTの増分

FOR～NEXTの増分

さきのFOR～NEXT文の場合、NEXT文で変数Iの値に＋1しました。

このようにNEXT文で＋1したのは、FOR文は正式には、つぎのような形になっているからです。

FOR I=1 TO 25 STEP 1

このSTEP 1の1を増分といい、「FOR文の変数の値を1ずつ増やしていけ」ということです。1ずつ増やすときは、増分の指定を省略することができるので、ふつうSTEP 1を指定しないだけです。

増分を2とか5にするときは、指定を省略することができませんから、FOR文の最後にSTEP 2、STEP 5を指定します。

増分を2と指定した場合は、NEXT文で変数の値に足す値は2となり、増分を5と指定した場合は5になります。

このように増分を指定すると、NEXT文で足す値が違ってきますから、初期値や最終値がおな

```
10 FOR I=1 TO 25 STEP 2
20 PRINT "♥";
30 NEXT
```

増分を2とすると繰り返す回数は13となる。

実行結果

```
run
♥♥♥♥♥♥♥♥♥♥♥♥♥
Ok
```

増分を2とすると、13個の♥を表示するようになる

増分を指定すると繰り返しの回数は減少

じであっても、FOR～NEXT文で繰り返す回数は違ってきます。

たとえば、162頁で示した♥を表示するプログラムの場合、初期値が1、最終値が25、増分は省略されていますから1です。したがってNEXTで＋1しますから、25個の♥が表示されましたが、おなじように初期値を1、最終値を25としても増分を2とすると、NEXT文で2を足すので、FOR文の変数Iの値は、

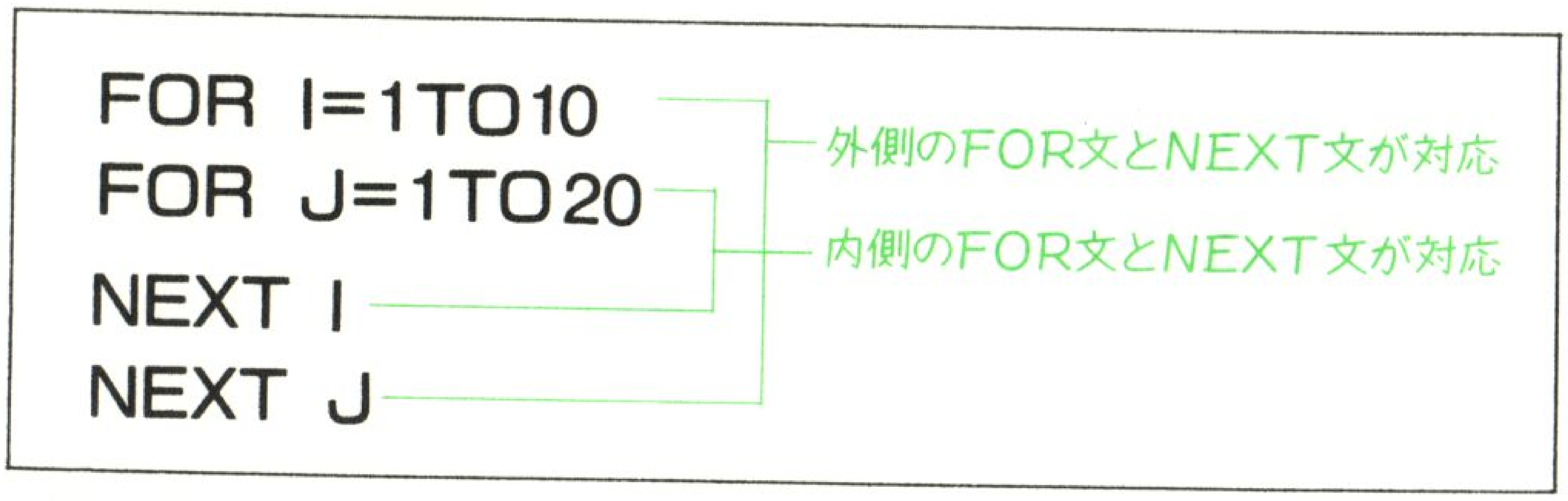
1　3　5　7　9　11　13　15　17
19　21　23　25

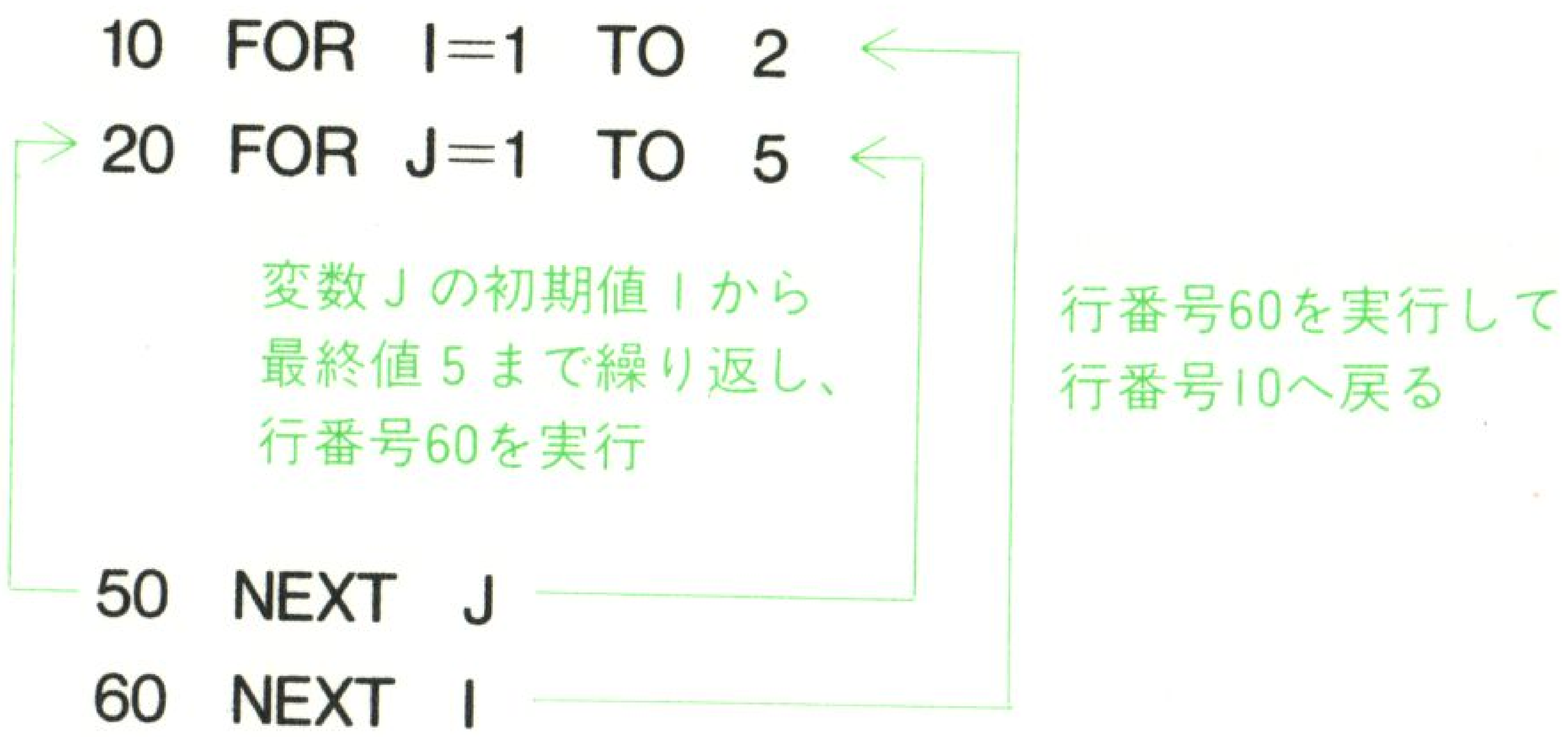
と変化していき、PRINT文を13回繰り返して、上に示すように、♥を13個表示すると繰り返しは終わります。

また増分を5とすると、NEXT文で5を足すので、FOR文の変数Iの値は、

1　6　11　16　21

と変化して、PRINT文を5回繰り返し、♥を5個表示すると終わりになります。





二重FOR～NEXT文

■二重FOR～NEXT

二重ＦＯＲ～ＮＥＸＴ（にじゅうフォア～ネクスト）文は、上に示すように、ＦＯＲ～ＮＥＸＴ文が二つ重なったものです。

このように、二つのＦＯＲ～ＮＥＸＴ文が重なった場合、外側のＦＯＲ文とＮＥＸＴ文が対応し、内側のＦＯＲ文とＮＥＸＴ文が対応します。

そして、外側のＦＯＲ文を実行すると、内側のＦＯＲ文を実行して、初期値から最終値まで、内側のＮＥＸＴ文と対応して繰り返します。繰り返しが終わると、外側のＮＥＸＴ文を実行して、外側のＦＯＲ文に戻ります。

このように、二重ＦＯＲ～ＮＥＸＴ文では、外側のＦＯＲ～ＮＥＸＴ文が１回実行されるたびに、内側のＦＯＲ～ＮＥＸＴ文が初期値から最終値まで繰り返されます。

二重ＦＯＲ～ＮＥＸＴ文の場合も、ＦＯＲ文で変数の値が最終値より大きいか比較し、ＮＥＸＴ文で変数の値に＋１して、繰り返しを行っていきます。

```
10 CLS:CONSOLE ,,0
20 FOR I=1 TO 30
30 R=INT(RND(1)*23)+1
40 FOR J=0 TO 77
50 LOCATE J,R
60 PRINT "▲"
70 BEEP 1:BEEP 0:BEEP 1:BEEP 0
80 FOR T=0 TO 200:NEXT
90 LOCATE J,R
100 PRINT "  "
110 NEXT
120 NEXT
```

- 10：画面クリア、ファンクションキーの表示を消す
- 20〜120：▲を移動させる回数
- 30：▲を表示させる位置を指定する乱数
- 40〜110：▲を画面左から右へ移動
- 50：▲を表示させる位置を指定
- 60：▲を表示
- 70：音を作る
- 80：ウエイト・ループ
- 90：空白を表示する位置を指定
- 100：空白を表示

▲を移動させるプログラム

上に二重ＦＯＲ~ＮＥＸＴを使ったプログラムを示します。

この場合、行番号20のＦＯＲ文と行番号120のＮＥＸＴ文が対応して働き、行番号40のＦＯＲ文と行番号110のＮＥＸＴ文とが対応して働きます。

行番号20のＦＯＲ文と行番号40のＦＯＲ文の間に、行番号30の乱数を作るステートメントがありますが、このように間にステートメントがあっても、二重ＦＯＲ~ＮＥＸＴ文はさきに示したように働きます。

では、プログラムの説明です。

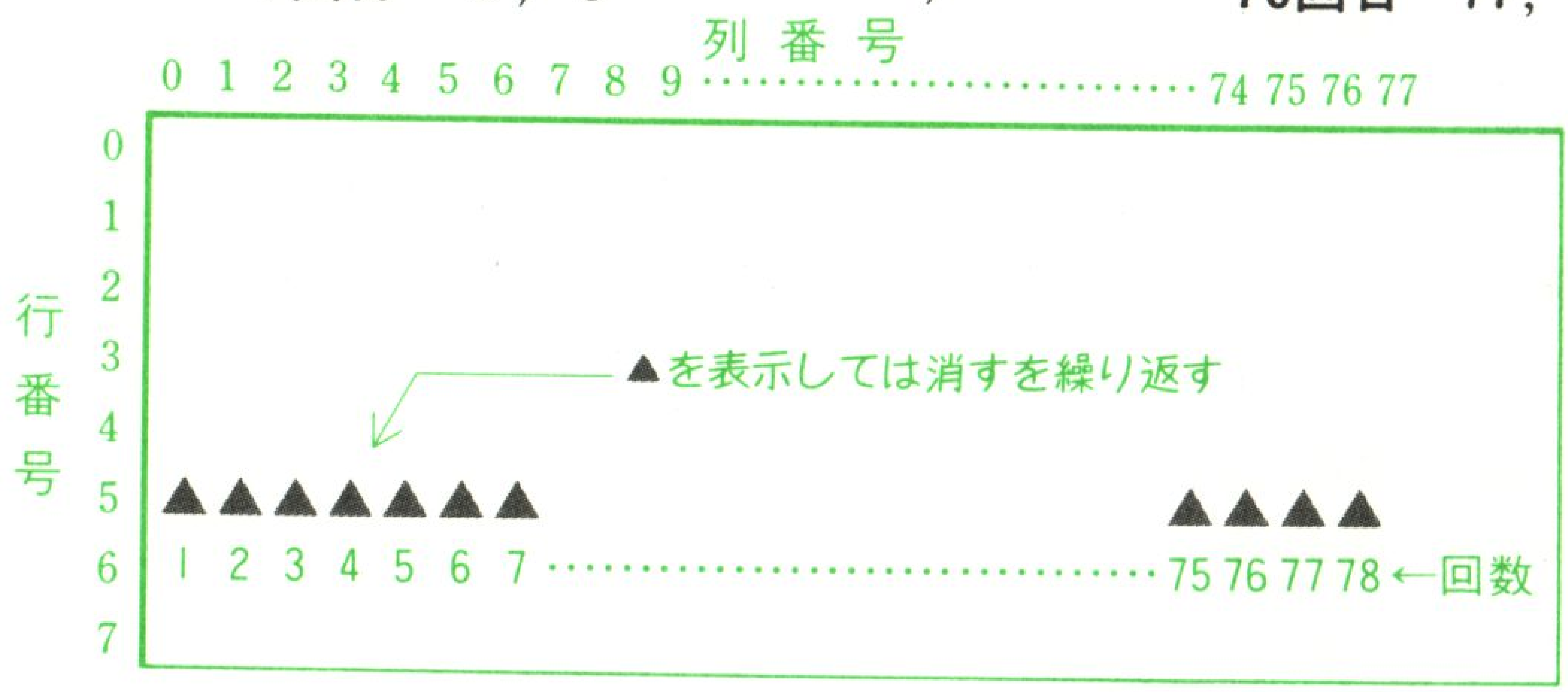
行番号10のＣＬＳ（クリアスクリーン）文で、テキスト画面に表示されている文字などを消去し、ＣＯＮＳＯＬＥ（コンソール）文で、ファンクションキーの内容の表示を消去します。

つぎに行番号20のＦＯＲ文を実行して、変数Ｉに初期値の1を代入します。

そして、行番号30の乱数を作るステートメントを実行して、そのときに作り出した乱数を変数Ｒに代入します。この変数Ｒに代入した乱数は、行

■LOCATE J, Rの変化（変数Rは5とする）

1回目	0, 5	5回目	4, 5	75回目	74, 5
2回目	1, 5	6回目	5, 5	76回目	75, 5
3回目	2, 5	7回目	6, 5	77回目	76, 5
4回目	3, 5	∫		78回目	77, 5



▲の移動の仕方

番号50と90のLOCATE文の変数Rに送られて▲と空白を表示する行番号の指定となります。

乱数を変数Rに代入すると、行番号40のFOR文を実行して、初期値の0から最終値の77まで、行番号110のNEXT文と対応して、その間にある行番号50から行番号100までのステートメントを繰り返します。繰り返す回数は78回です。

もちろん、単に78回繰り返すわけではなく、繰り返すたびに、行番号40のFOR文の変数Jの初期値0から最終値77までを、行番号50と90のLOCATE文の変数Jに送って、▲と空白を表示する列番号を指定します。

```
50 LOCATE J, R
```
↑ ▲を表示する位置を指定

変数Rには、行番号30で作り出された乱数が送られてきます。たとえば、変数Rの乱数が5であるとすると、繰り返しの1回目は、FOR文の変数Jの初期値0が送られてきますから、行番号50のLOCATE文は、

```
50 LOCATE 0, 5
```

となって、その位置に、行番号60のPRINT文

で▲を表示します。行番号70と80は、あとで示します。

▲を表示すると、行番号90のLOCATE文を実行します。LOCATE文の変数JとRには、行番号50のLOCATE文の変数JとRとおなじ値が送られてきますから、

90 LOCATE 0, 5

となって、その位置に、行番号100のPRINT文で空白を表示します。つまり、▲とおなじ位置に空白を表示するわけですから、表示されている▲が消されることになります。

▲などを移動させるには、このように▲を表示したら、その上に空白を表示して消すということを繰り返すわけです。

空白を表示して▲を消したら、行番号110のNEXT文を実行して、行番号40のFOR文に戻ります。

こんどは、FOR文の変数Jは1ですから、その1が、行番号50のLOCATE文の変数Jに送られてきますから、

50 LOCATE 1, 5

となって、その位置に行番号60のPRINT文で▲を表示します。

また、行番号90のLOCATE文にもおなじ値が送られてきますから、

90 LOCATE 1, 5

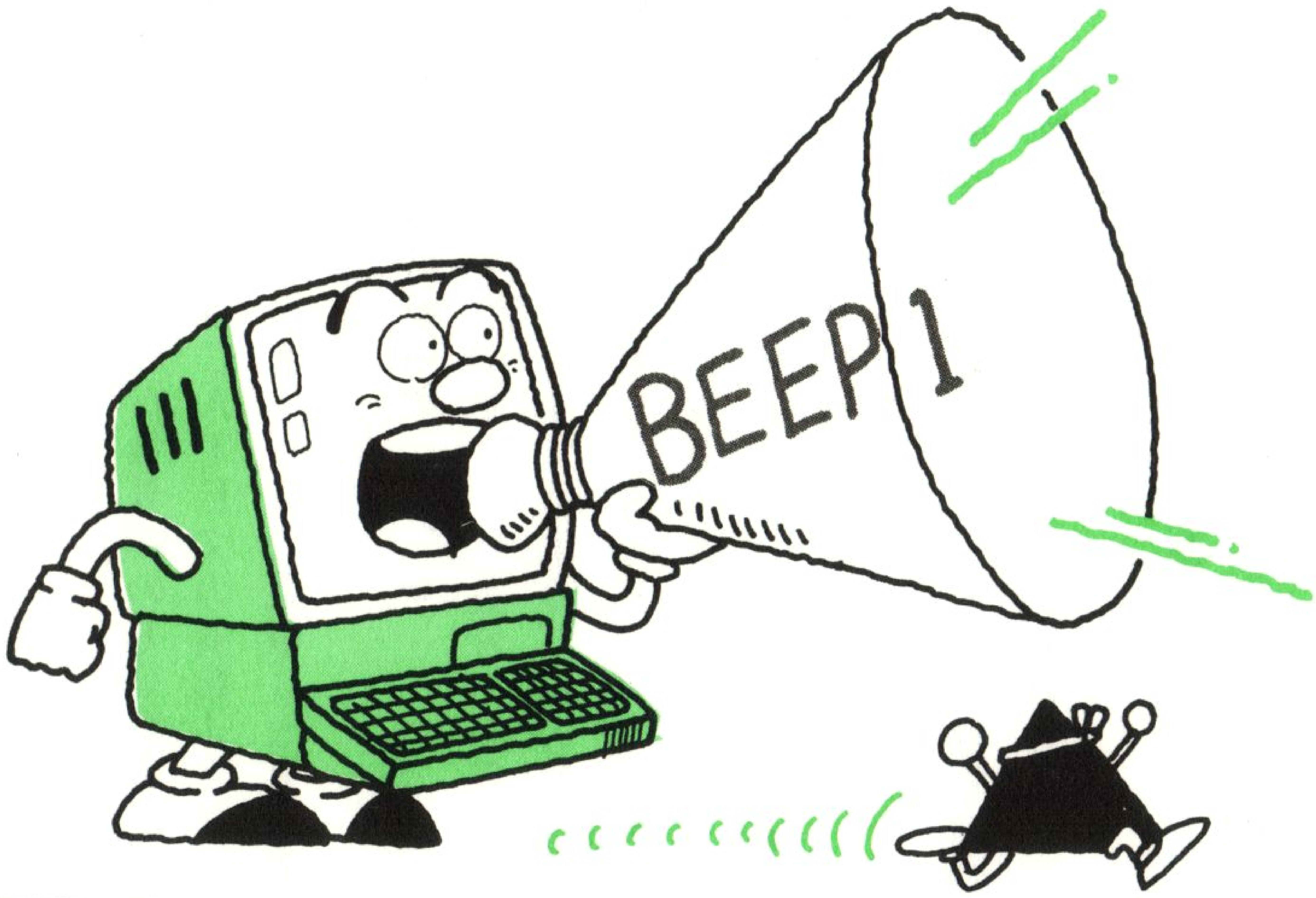
となって、その位置に行番号100のPRINT文で空白を表示します。

このことを、行番号40のFOR文の最終値77まで繰り返すと、▲はまえの頁に示すように、行番号5の位置を、左端から右端まで移動します。

最終値77で▲が右端に移動したら、行番号40のFOR～NEXT文の繰り返しは終わって、行番号120のNEXT文を実行して、行番号20のFOR文に戻ります。

そして、行番号30で乱数を作るステートメントを実行して、そのとき作り出した乱数を変数Rに代入し、行番号40のFOR文の実行に移ります。

このあとのことは、まえに示したこととおなじですから省略します。



BEEP文は音を出す

行番号20のFOR文の最終値30をこえると、▲の移動は終了となります。

つまり、このプログラムの二重FOR～NEXT文の場合、行番号40と110のFOR～NEXT文で▲を画面の左から右に向けて移動させ、行番号20と120のFOR～NEXT文で、▲の移動の回数を決めています。

さて、行番号70と80のステートメントです。行番号80のFOR～NEXT文は、ウエイト・ループです。▲を表示したあと、すぐに空白を表示すると▲の移動がわからないので、空白を表示するステートメントの実行を遅らせています。

行番号70のBEEP（ビープ）文で、▲の移動の際の音を作っています。

BEEP文は、音を出すステートメントで、つぎのように1と0を指定して使います。

BEEP　1＝音を鳴りっぱなしにする
BEEP　0＝音を止める

もちろん、つぎのようにしても、音を出します。

BEEP＝音を出す

INPUT "プロンプト";〈変数〉,〈変数〉,………

"プロンプト"　＝どんなデータを入力するのかを示す案内の文を「"」で囲んで指定。
プロンプトの指定は省略できる。

〈変数〉　＝数値を入力するときは数値変数を指定。
文字列を入力するときは文字変数を指定。

"プロンプト";　＝プロンプトのあとに「;」セミコロンをつけるとプロンプトと?マークを表示。

"プロンプト",　＝プロンプトのあとに「,」コンマをつけると、プロンプトを表示して?マークは表示されない。

INPUT文

INPUTはデータの入力

プログラムでデータを処理させるには、あらかじめ処理させるデータをプログラムのなかに書いておく方法と、プログラムを実行させてから、キーボードのキーを叩いてデータを入力する方法の二つがあります。

INPUT（インプット）文は、プログラムを実行させたあとデータを入力するステートメントで、INPUT文を実行すると、?マークを表示してデータの入力を求めてきます。

ところで、INPUT文は上に示すような形をしています。

〈プロンプト〉のところには、どんなデータを入力するのか、入力するデータの案内をするガイド文を「"」引用符で囲んで指定します。

INPUT "スウチ　ノ　ニュウリョク"

このように「"」引用符で囲むのを忘れると、プロンプトはメモリに記憶されませんから、表示

①
```
run
?■
```
プロンプトを「″」で囲むのを忘れた場合

②
```
run
スウチ ノ ニュウリョク?■
```
プロンプトのあとに「；」をつけた場合

③
```
run
スウチ ノ ニュウリョク■
```
プロンプトのあとに「，」をつけた場合

④
```
run
?■
```
プロンプトの指定を省略した場合

プロンプトや「；」などの指定の仕方による表示

されずに、？マークだけが表示される（①参照）ことになりますから、注意してください。

また、プロンプトを指定したあとに「；」セミコロンをつけると、②に示すようにプロンプトと？マークが表示されますが、「，」コンマをつけるとプロンプトだけが表示されて、？マークは表示されないようになります（③参照）。

プロンプトは、入力するデータを間違えないようにするためにつけておきますが、必ず指定しなければならないというものではなく、省略することができます。プロンプトの指定を省略すると？マークだけが表示されます（④参照）。

〈変数〉のところには、数値変数あるいは文字変数を指定します。数値を入力するときは数値変数、文字列を入力するときは文字変数です。

〈変数〉は、必要なだけ指定することができますが、複数個の〈変数〉を指定するときは、〈変数〉と〈変数〉の間を「，」コンマで区切ります。

〈変数〉と〈変数〉の間を「，」で区切らないと、一つの変数とみなされることになります。

```
10 INPUT "ﾀｽ ｶｽﾞ ｦ ﾆｭｳﾘｮｸ";A
20 B=B+A
30 PRINT "ｺﾞｳｹｲ";B
40 GOTO 10
```

実行結果

```
run ■                      ←RUNコマンドを入力して実行させる
ﾀｽ ｶｽﾞ ｦ ﾆｭｳﾘｮｸ? 150⏎     ←150を入力して⏎キーを叩く
ｺﾞｳｹｲ 150                  ←合計を表示
ﾀｽ ｶｽﾞ ｦ ﾆｭｳﾘｮｸ? 200⏎     ←200を入力して⏎キーを叩く
ｺﾞｳｹｲ 350                  ←合計を表示
ﾀｽ ｶｽﾞ ｦ ﾆｭｳﾘｮｸ?■
```

INPUT文を使ったプログラム

■INPUTでデータを入力

INPUT文によるデータの入力です。

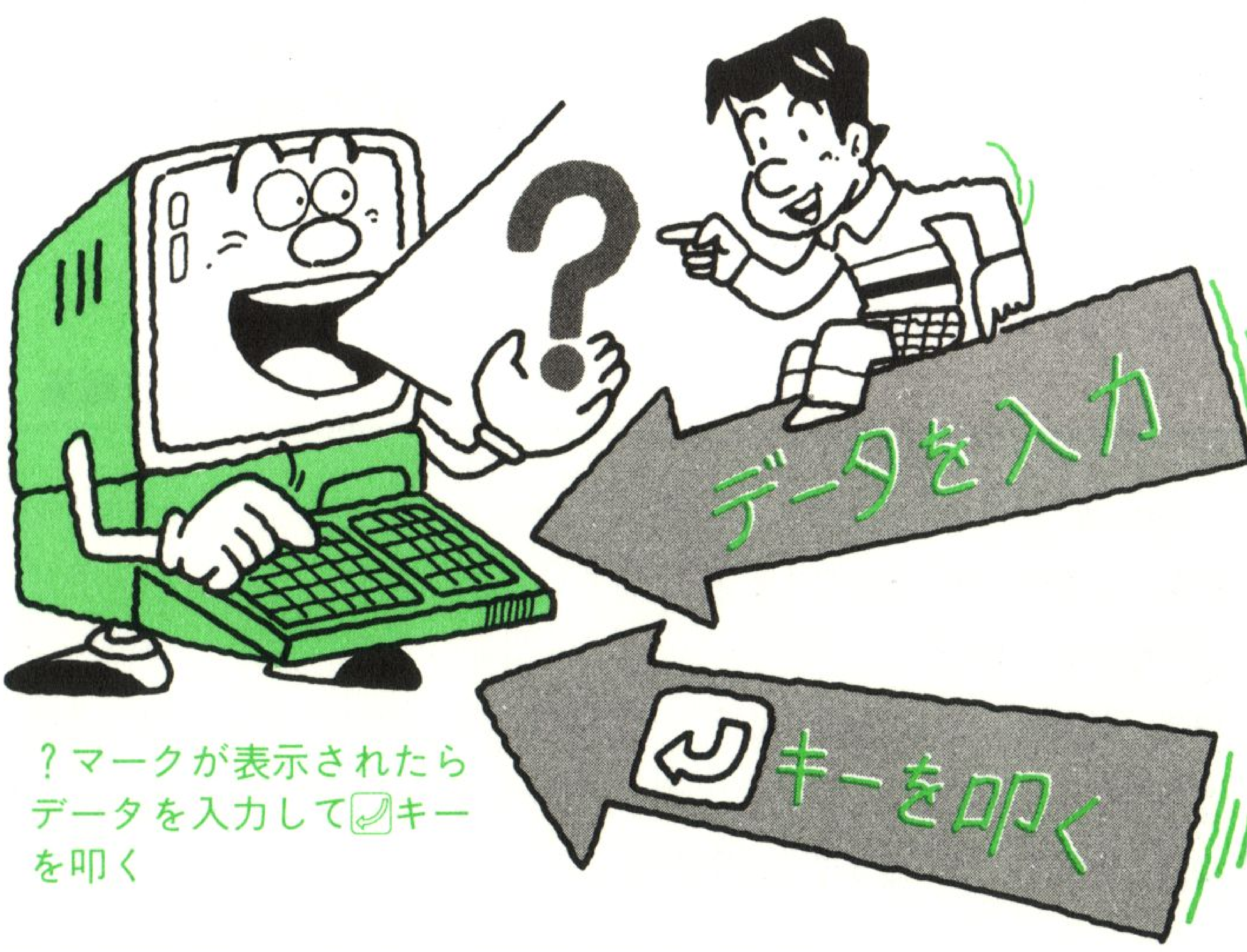
上に、INPUT文を使ったプログラムを示します。このプログラムは、入力したデータの合計を求めるというものです。

このプログラムを入力して実行させると、行番号10のINPUT文が実行されて、上に示すように、プロンプトと?マークが表示されて、データの入力を求めてきます。データを入力しないかぎり、この状態がつづきます。これをキー入力待ちといいます。

そこで、150を入力することにします。テンキーを叩いて150と入力したら⏎キーを叩きます。これで150は、変数Aに代入されます。

これがINPUT文によるデータの入力です。ついでに残りのステートメントの働きを示しておくことにします。

150を入力すると、つぎに行番号20のB＝B＋Aが実行されます。変数AにはINPUT文で150を入力しましたから、変数Aは150、変数Bにはなん



?マークが表示されたらデータを入力して⏎キーを叩く

INPUT文によるデータの入力

の値も代入していませんから、変数Bは０です。したがって150＋０が行われて、その結果の150が＝の左側の変数Bに代入されます。

つぎに行番号30のＰＲＩＮＴ文が実行されて、変数Ｂの内容、つまり、変数Ｂに代入した計算の結果150を表示します。

計算の結果を表示すると、行番号40のＧＯＴＯ文を実行して行番号10にジャンプしますから、ふたたびＩＮＰＵＴ文が実行されて、プロンプトと?マークを表示してデータの入力を求めてきます。

こんどは、200を入力することにして、テンキーを叩いて200を入力したら⏎キーを叩きます。すると、200が変数Ａに代入されます。

このように新たに200を代入すると、まえに代入した150は消えて、変数Ａは200になります。

つぎに行番号20のＢ＝Ｂ＋Ａが実行されると、変数Ａには200、変数Ｂには計算結果の150が代入されていますから、150＋200が行われて、その結果の350が変数Ｂに代入されます。この場合もまえに変数Ｂに代入した150は消えて、変数Ｂの値は新たに

```
10 INPUT "ﾃﾞｰﾀ ﾊ 4ｺ";A,B,C,D
20 HEIKIN=(A+B+C+D)/4
30 PRINT "ﾍｲｷﾝﾁ";HEIKIN
40 GOTO 10
```

実行結果

```
run
ﾃﾞｰﾀ ﾊ 4ｺ? 98,78,69,89⏎
ﾍｲｷﾝﾁ 83.5
ﾃﾞｰﾀ ﾊ 4ｺ?■
```

← コンマで区切って4個のデータを入力したら⏎キーを叩く

← 平均値を求めて表示

複数個のデータの入力

代入された350となって、PRINT文で計算の結果350を表示します。

INPUTで複数個のデータを入力

複数個のデータの入力です。上にプログラムを示します。このプログラムは、4個のデータの平均値を求めるものなので、4個のデータを入力できるように、INPUT文にA、B、C、Dの4個の数値変数を並べています。

プログラムを入力して実行させると、プロンプトと?マークが表示されますから、4個のデータを入力します。

4個のデータを入力するには、4個のデータの間を「，」コンマで区切ります。「，」コンマで区切らないと、一つのデータとみなされます。

この場合、98、78、69、89と入力します。入力して⏎キーを叩くと、4個のデータは、最初から順にA、B、C、Dの4個の変数に代入されます。

98 → A
78 → B
69 → C
89 → D

```
run
デ゛ータ ハ 4コ? 98,78,69,89
ヘイキンチ 83.5
デ゛ータ ハ 4コ? 100,99,65
?Redo from start
デ゛ータ ハ 4コ?■
```

（100,99,65の後 ← データの数が少ない）
（?Redo from start／データ ハ 4コ? ← ふたたびデータの入力を求めてくる）

```
run
デ゛ータ ハ 4コ? 98,78,69,89
ヘイキンチ 83.5
デ゛ータ ハ 4コ? 100,98,99,100,68
?Redo from start
デ゛ータ ハ 4コ?■
```

（100,98,99,100,68の後 ← データの数が多い）
（?Redo from start／データ ハ 4コ? ← ふたたびデータの数を求めてくる）

変数の数に応じてデータを入力すること

この場合、ＩＮＰＵＴ文に並んでいる変数の数より少ないデータを入力したり、また、変数の数より多いデータを入力したりすると、?Redo from start（初めから入力しなおしてください）というメッセージが表示されて、ふたたびプロンプトと?マークが表示されます。

4個のデータを入力すると、行番号20の計算式が実行されます。計算式にカッコで括られた部分があると、そのなかの計算が最優先に実行されますから、変数Ａ、Ｂ、Ｃ、Ｄに代入した98、78、69、89が足され、その結果を4で割ります。計算の結果、求められた平均値は＝の左にある変数ＨＥＩＫＩＮに代入されます。

つぎに行番号30のＰＲＩＮＴ文が実行されて、変数ＨＥＩＫＩＮに代入された計算の結果を表示し、つぎの行番号40のＧＯＴＯ文で行番号10にジャンプします。そして、ふたたびＩＮＰＵＴ文が実行されて、データの入力を求めてきます。

ここで示したプログラムを終了させるときは、STOPキーを押します。

IF〈条件式〉THEN　行番号　あるいは文

〈条件式〉　＝条件を設定する

THEN　＝条件が成立したとき行うことを設定する

THEN　行番号＝条件が成立したとき、指定された行番号にジャンプして、その行番号のステートメントから実行

THEN　文　＝条件が成立したとき文を実行。たとえば THEN PRINT 〝アタリ〟なら、PRINT文でアタリと表示する

IF～THEN文

IF～THENは条件に基づいて実行

プログラムは、行番号の小さいほうから大きいほうへと実行されていきます。この流れを変えるステートメントとしては、156頁で示したGOTO文と、IF～THEN文があります。ただし流れを変えるといっても、GOTO文は指定した行番号に無条件にジャンプして流れを変えてしまいますが、IF～THEN文は、もし（IF）天気ならば（THEN）ゴルフに行くというように、天気という条件によって流れを変えます。

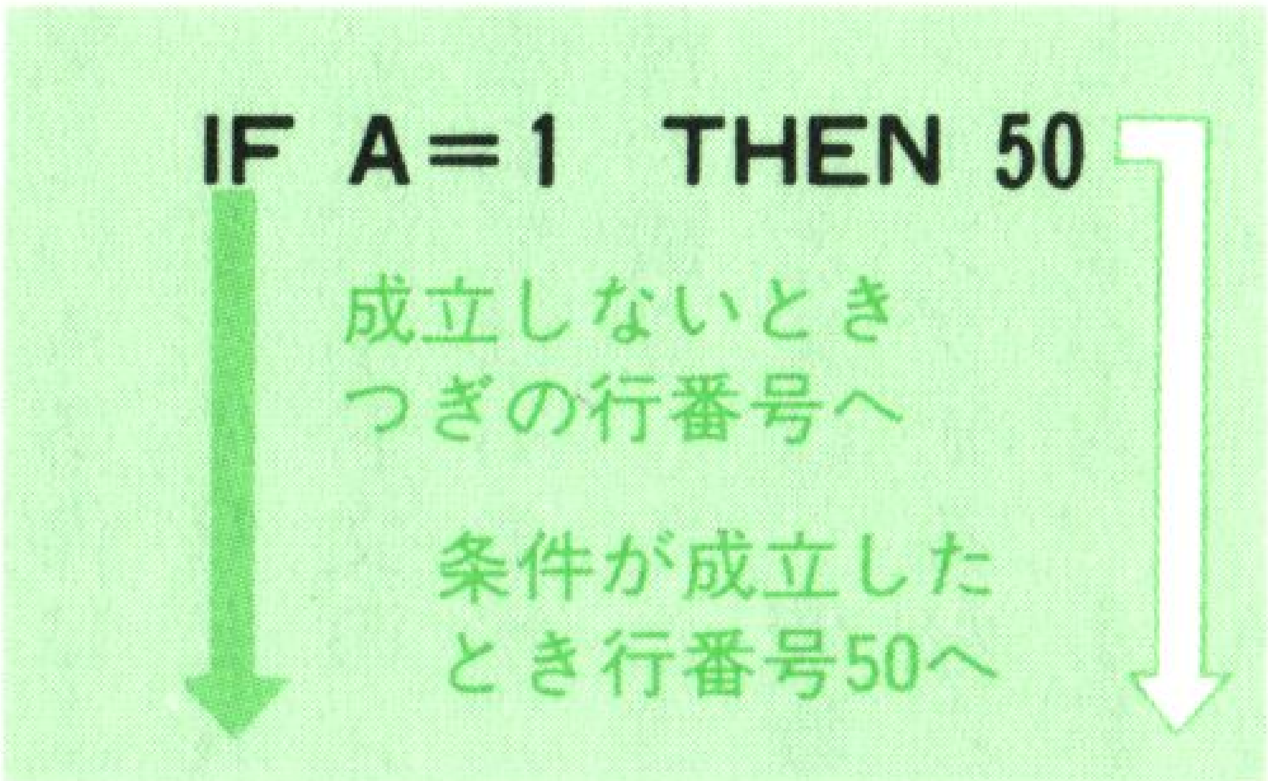
IF～THEN文は、上に示すような形をしていて、IF文の〈条件式〉のところには、条件式をあてはめます。そしてTHEN文には、この条件が成立したとき行うことを指定します。

たとえば、変数Aの値が1のとき、行番号50にジャンプするなら、条件は変数Aが1と等しいことですから、IF文の条件式にA＝1と指定します。そして、A＝1の条件が成立したとき行番号

関係演算子	条件式
① ＝（等しい）	A＝B　（AがBと等しいこと）
② ＜（小さい）	A＜B　（AがBより小さいこと）
③ ＞（大きい）	A＞B　（AがBより大きいこと）
④ ＜＝（等しいか小さい）	A＜＝B（AがBと等しいか小さいこと）
⑤ ＞＝（等しいか大きい）	A＞＝B（AがBと等しいか大きいこと）
⑥ ＜＞（等しくない）	A＜＞B（AがBと等しくないこと）

```
10 INPUT B                  ←データを入力
20 IF B>50 THEN 50          ←変数Bの値が50より大きいとき、行番号50にジャンプ
30 PRINT "ooooooooo"        ←B<50のとき実行
40 GOTO 10
50 PRINT "xxxxxxxxx"        ←B>50のときのジャンプしてきて実行
60 GOTO 10
```

IF文の条件式とIF文を使ったプログラム



50にジャンプするわけですから、上に示すように、THEN文のあとに行番号50を指定します。

これで変数Aが1のとき、行番号50にジャンプし、Aが1でないときは、つぎの行番号のステートメントを実行します。

上のIF~THEN文を使った簡単なプログラムは、行番号10のINPUT文で変数Bに値を入力し、その値が50より大きいとき、行番号20のIF文の条件が成立して、THEN 50で行番号50にジャンプして、PRINT文で×を表示します。変数Bに入力した値が50より小さいと、IF文の条件は成立しませんから、つぎの行番号30のPRINT文で○を表示します。

IF文の条件は、条件式を見ただけではわからないので、行番号20のIF文の条件に、いろいろな条件式をあてはめて試してください。

```
10 INPUT "スウチ データ";A
20 IF A=99999 THEN END
30 B=B+A
40 PRINT "ゴウケイ";B
50 GOTO 10
```

99999を入力

A＝99999の条件が成立すると、THEN文のENDを実行して終わる

```
10 INPUT "データ ノ ニュウリョク";A
20 IF A=0000 THEN 50
30 B=B+A
40 GOTO 10
50 PRINT "ゴウケイ";B
```

0000を入力

A＝0000の条件が成立すると行番号50にジャンプ

合計を表示して終わる

無限ループから抜け出すにはIF～THEN文で

無限ループから抜け出すには

このプログラムもSTOPキーを押さないとプログラムは終了しません。これは、GOTO文で行番号10に戻して繰り返しているからです。

FOR～NEXT文の場合は、繰り返す回数を決めますから、決められた回数だけ繰り返すと終わりますが、GOTO文の場合は、繰り返す回数の指定がないので無限に繰り返すため、GOTO文での繰り返しを、**無限ループ**といいます。

GOTO文の無限ループから抜け出す（繰り返しを終了する）にはIF～THEN文を使います。

たとえばGOTO文で繰り返して、INPUT文で変数Aにデータを入力しているとすれば、上に示すように、IF文に変数Aの値が99999と等しいことという条件を設定して、INPUT文で変数Aに99999を入力したとき、THEN文を実行させて終わらせるようにします。

もちろん99999ではなくて、0000を入力したときとしてもかまいません。データとして入力しない値で条件を設定すればよいのです。

10
20
30
40 ← **メインルーチン**（本筋のプログラム）
50
60
70 ← **END（エンド）文**（メインルーチンの最後の行番号に置く。ないとエラーになる）
100
110
120 ← **サブルーチン**（繰り返しでてくる手順を一つにまとめて本筋のプログラムから切り離して置く）
130
140

メインルーチンとサブルーチン

GOSUBはサブルーチンにジャンプ RETURNはメインルーチンに戻る

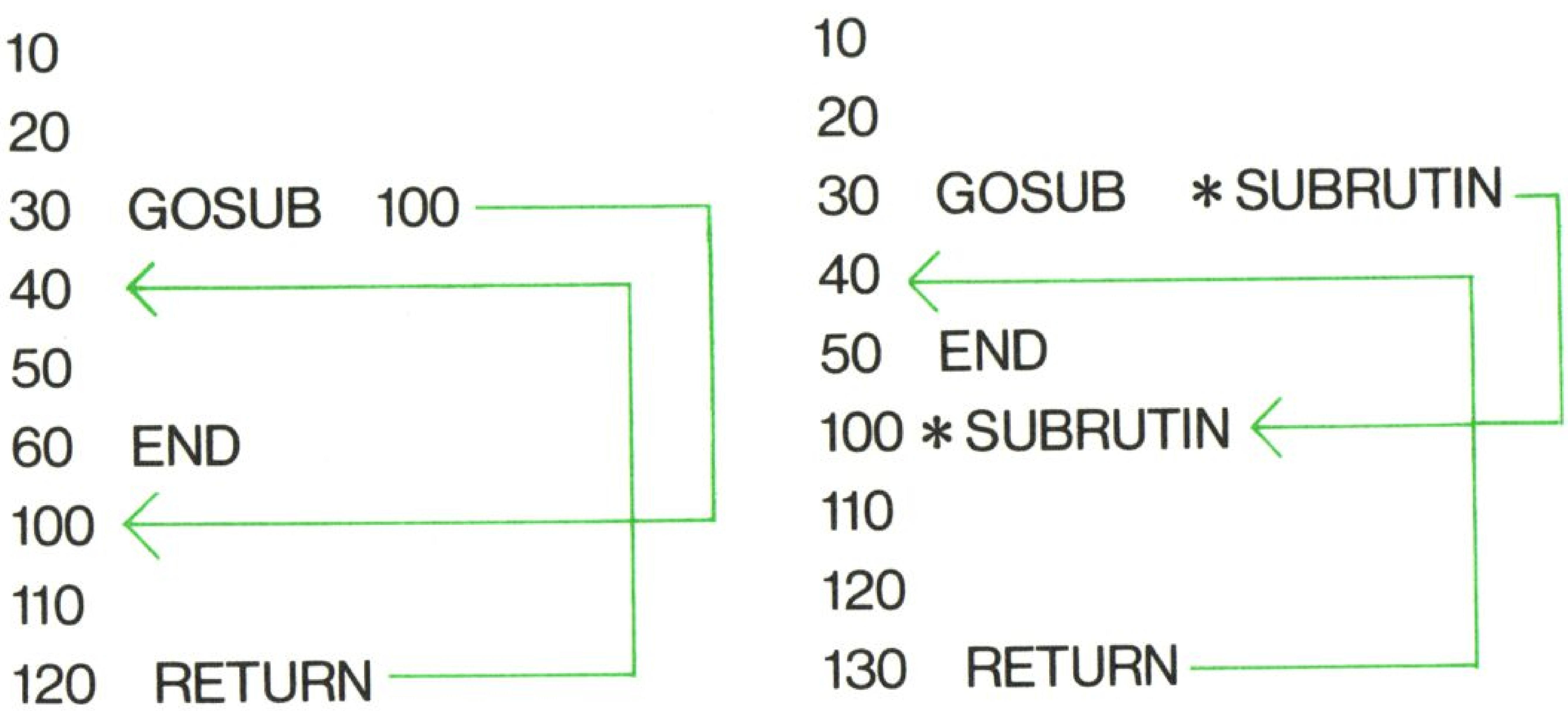
プログラムを作っていると、同じ手順が繰り返してでてくることがありますが、同じ手順をいくつも書いていたのではプログラムが長くなるので、同じ手順を一つにまとめて、本筋のプログラムから切り離します。そして、本筋のプログラムでその手順が必要になったとき、その手順にジャンプさせて処理するようにします。この場合、本筋のプログラムを**メインルーチン**、本筋のプログラムから切り離した手順を**サブルーチン**といいます。

サブルーチンは、メインルーチンのあとにおきますが、メインルーチンの最後の行番号にＥＮＤ文を置いて、そのあとに置きます。ＥＮＤ文がないとエラーになるので注意してください。

そして、メインルーチンでサブルーチンの手順を必要とするところから、サブルーチンにジャンプさせて、その手順を実行させます。

GOSUB 〈行番号〉 ― メインルーチンでサブルーチンの手順を必要としたところにおく

RETURN ― サブルーチンの最後の行番号におく



GOSUB～RETURN文

その場合に使うのが、ＧＯＳＵＢ～ＲＥＴＵＲＮ文です。

ＧＯＳＵＢ～ＲＥＴＵＲＮ文は、ＧＯＳＵＢ文とＲＥＴＵＲＮ文の二つにわかれていて、ＧＯＳＵＢ文はメインルーチンでサブルーチンの手順を必要とするところに置きます。ＧＯＳＵＢの〈行番号〉のところには、サブルーチンの先頭の行番号を指定します。

またＲＥＴＵＲＮ文は、サブルーチンの最後の行番号に置きます。このようにすると、サブルーチンの手順を実行し終わったあと、ＧＯＳＵＢ文でサブルーチンにジャンプした、つぎの行番号に戻ってきます。

ＧＯＳＵＢ文の〈行番号〉のところには、サブルーチンの先頭の行番号ではなく、サブルーチン名を指定することができますが、その場合は、サブルーチン名のまえに＊記号を指定します。

たとえば、ＧＯＳＵＢ文の〈行番号〉のところに、ＳＵＢＲＵＴＩＮと指定するとすれば、つぎのようになります。

```
10 TEIHEN=24              ← 底辺の値を代入
20 TAKASA=32              ← 高さの値を代入
30 GOSUB *MENSEKI         ← サブルーチンMENSEKIにジャンプ
40 TEIHEN=47              ← 底辺の値を代入
50 TAKASA=19              ← 高さの値を代入
60 GOSUB *MENSEKI         ← サブルーチンMENSEKIにジャンプ
70 END
80 *MENSEKI               ↓ 底辺×高さ÷2の計算
90 SANKAKU=TEIHEN*TAKASA/2
100 PRINT "ｻﾝｶｸｹｲ ﾉ ﾒﾝｾｷ ﾊ";SANKAKU
110 RETURN                ↑ 三角形の面積を表示
```

GOSUB~RETURN文を使ったプログラム

GOSUB ＊SUBRUTIN

このようにサブルーチン名を指定したときは、サブルーチンの先頭の行番号のところにも、おなじように＊記号をつけて、

＊SUBRUTIN

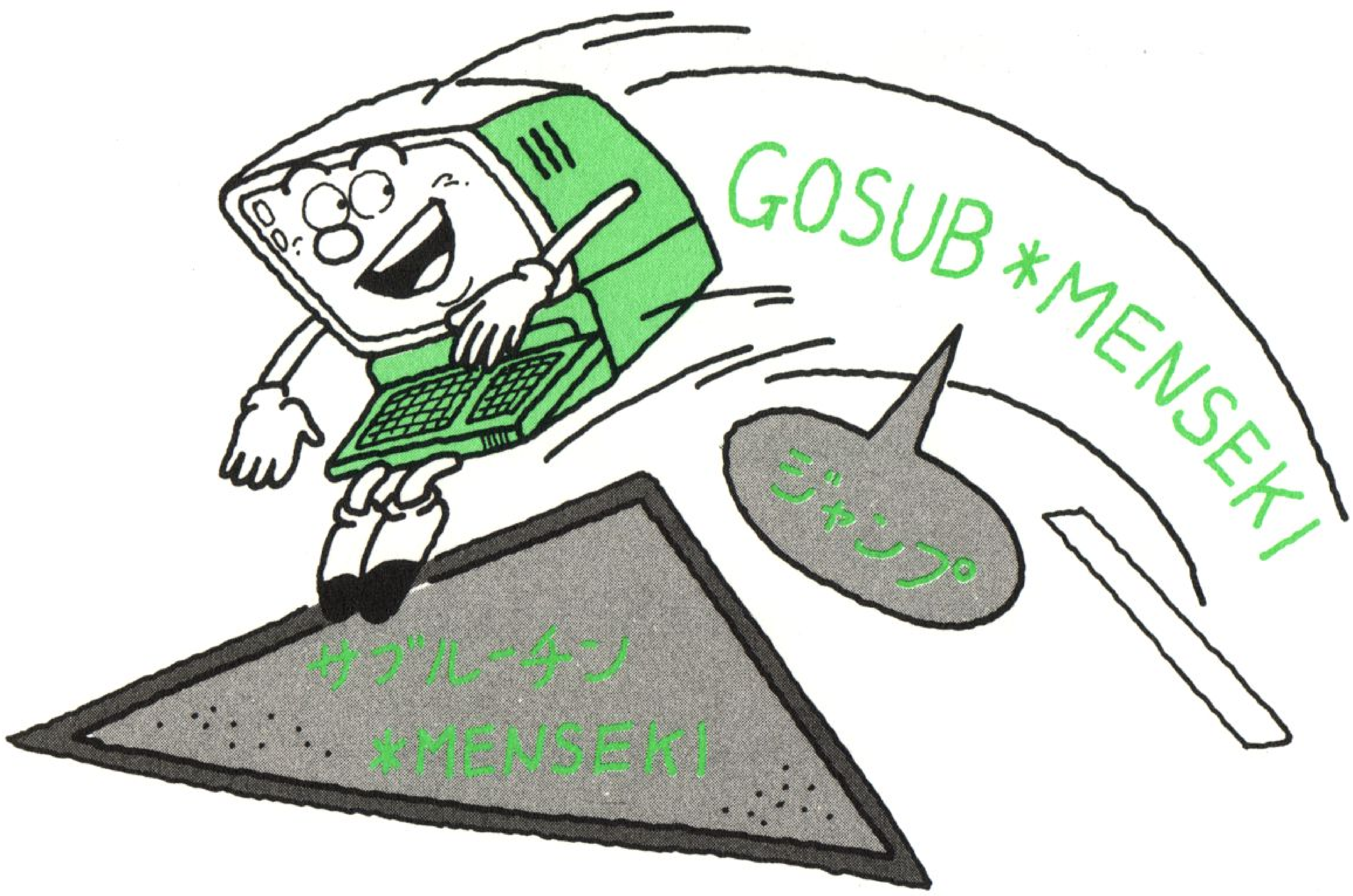
サブルーチン名を指定します。

GOSUB~RETURN文を使った簡単なプログラムを示します。

このプログラムは三角形の面積を求めるもので、行番号10から行番号70までがメインルーチン、行番号80から110までがサブルーチンです。

行番号10で、底辺の値24を変数TEIHENに代入します。行番号20で、高さの値32を変数TAKASAに代入します。

底辺と高さの値を変数に代入したら、つぎに三角形の面積を計算する手順が必要ですから、行番号30にGOSUB文を置いて、三角形の面積を計算するサブルーチンにジャンプさせます。このプログラムのGOSUB文には、サブルーチン名＊MENSEKIが指定されていますから、サブル



GOSUB　＊MENSEKI

ーチン＊ＭＥＮＳＥＫＩにジャンプします。
サブルーチン＊ＭＥＮＳＥＫＩにジャンプしてくると、行番号90の三角形の面積を計算する底辺×高さ÷2を行い、その結果を＝の左にある変数ＳＡＮＫＡＫＵに代入します。

そして、行番号100のＰＲＩＮＴ文で変数ＳＡＮＫＡＫＵに代入された計算の結果、つまり三角形の面積384が表示されます。

三角形の面積を表示すると、行番号110のＲＥＴＵＲＮ文で、ＧＯＳＵＢ文でサブルーチンにジャンプした、つぎの行番号40に戻ります。

行番号40に戻ると、もう一つの三角形の面積を求めるために、行番号40のステートメントで、底辺の値47を変数ＴＥＩＨＥＮに代入します。また行番号50のステートメントで、高さの値19を変数ＴＡＫＡＳＡに代入します。

そしてさきの場合と同じように、行番号60のＧＯＳＵＢ文で、サブルーチン＊ＭＥＮＳＥＫＩにジャンプして、三角形の面積を求め、その結果の446・5を表示して戻ります。

READ〈変数〉,〈変数〉,〈変数〉, ……

ＤＡＴＡ文においたデータを変数に読み取る

DATA〈データ〉,〈データ〉, ……

ＲＥＡＤ文で変数に読み取るデータをおく

〈変　数〉＝ＤＡＴＡ文においたデータが数値のときは数値変数、文字列のときは文字変数を指定

〈データ〉＝データには数値あるいは文字列を指定

READ～DATA文

READ~DATA文はプログラムに書かれたデータを読み取る

ＲＥＡＤ~ＤＡＴＡ文は、あらかじめプログラムのなかにデータを書いておき、プログラムを実行させると、そのデータを変数に読み取るというステートメントです。

ＲＥＡＤ~ＤＡＴＡ文は、ＲＥＡＤ文とＤＡＴＡ文の二つにわかれていて、ＲＥＡＤ文の〈変数〉のところには、ＤＡＴＡ文のデータを読み取る変数を指定します。データが数値のときは数値変数、文字列のときは文字変数です。

ＤＡＴＡ文には、ＲＥＡＤ文で変数に読み取らせるデータを必要なだけ指定します。データとしては、数値はもちろん、文字列もおくことができます。

ＲＥＡＤ文の変数にＤＡＴＡ文のデータが読み取られる様子はつぎのとおりです。

たとえば、変数Ａに10、20、30の3個のデータ

```
10 FOR I=1 TO 3
20 READ A
30 PRINT "A ノ デ゛ータ";A
40 NEXT
50 DATA 10,20,30
```

READ文を3回繰り返す（10〜40行）
変数Aにデータを読み取る（20行）
変数Aに読み取るデータ（50行）

実行結果

```
run
A ノ デ゛ータ 10
A ノ デ゛ータ 20
A ノ デ゛ータ 30
Ok
```

1回目に読み取られたデータ
2回目に読み取られたデータ
3回目に読み取られたデータ

READ文で1個の変数へデータを読み取る

を読み取らせるには、DATA文に10、20、30の3個のデータをおきます。そして、READ文に変数Aを指定します。

READ文を1回実行すると、DATA文におかれたデータの一つを変数Aに読み取りますから、3個のデータを読み取らせるには、READ文を3回実行させなければなりません。

そこで、READ文をFOR~NEXT文で囲み、3回繰り返してREAD文を実行させると、変数Aに3個のデータが読み取られます。

このプログラムでは、行番号30のPRINT文で変数Aの内容を表示させていますから、変数Aに読み取った順にデータが表示されていきます。

つぎは、複数個の変数へのデータの読み取りです。変数A、B、Cの3個の変数に10から90までの9個のデータを読み取らせるには、DATA文に10から90までの9個のデータをおき、READ文には、変数A、B、Cを指定します。

READ文を一回実行すると、最初のデータ10を変数Aに、二番目のデータ20を変数Bに、三番

```
10 FOR I=1 TO 3
20 READ A,B,C
30 PRINT "A B C ノ デ゛ータ";A;B;C
40 NEXT
50 DATA 10,20,30,40,50
60 DATA 60,70,80,90
```

READ文を3回繰り返す（行10～40）

変数A，B，Cに読み取られるデータ（行50～60）

```
run
A B C ノ デ゛ータ 10   20   30
A B C ノ デ゛ータ 40   50   60
A B C ノ デ゛ータ 70   80   90
Ok
```

（1回目に読み取られたデータ／2回目に読み取られたデータ／3回目に読み取られたデータ）

READ文で複数個の変数へデータを読み取る

目のデータ30を変数Cに読み取ります。

9個のデータをREAD文で3個の変数に読み取らせるには、FOR～NEXT文で囲んで、三回繰り返して実行させればよいわけです。

このプログラムでも、行番号30のPRINT文で変数A、B、Cの内容を表示させていますから、READ文で変数A、B、Cに読み取られるたびに、読み取られたデータが表示されていきます。

ENDは実行の終了を知らせる

END文はパソコンに、プログラムの実行の終了を知らせるステートメントで、END文を実行した時点で、パソコンはプログラムの実行を終了してOKを表示します。

END文は、終わりという意味であることから、プログラムの最後におくものと考えられがちですが、END文は、プログラムの実行を終了させるステートメントですから、プログラムのどこにおいてもかまいません。

```
10 GOSUB 500
20 GOSUB 600
500 LOCATE 10,10
510 PRINT "xxxxxx"
520 RETURN
600 LOCATE 10,20
610 PRINT "oooooo"
620 RETURN
```

メインルーチン（10〜20）
END文がない（20の後）
サブルーチン（500〜520）
サブルーチン（600〜620）

```
run          xxxxxx
RETURN without GOSUB in 520
Ok
```

メインルーチンの最後にENDがないのでエラーになる

END文がないとエラーになる（GOSUB～RETURN）

なお、GOSUB～RETURN文を使ったプログラムの場合は、メインルーチンの最後に必ずEND文をおくことになっています。このEND文を忘れると、上のプログラムに示すようにエラーになりますから注意してください。

ただし、メインルーチンの最後にGOTO文を置いて繰り返していたり、メインルーチンをFOR～NEXT文で繰り返している場合は、END文は必要ありません。

AUTOは行番号を自動的につける

プログラムを入力するときに、一つ一つ行番号を入力していたのでは面倒という場合は、AUTOコマンドを使うと便利です。

AUTOは、行番号を自動的につけてくれるコマンドで、ファンクションキーの[f・2]キーを押すと、AUTOと表示されます。

[f・2]＝AUTOコマンド

AUTOと表示されたら[⏎]キーを叩きます。こ

AUTO〈行番号〉,〈増分〉

①指定を省略した場合

```
auto⏎
10 print⏎
20 print⏎
30■
```

行番号が10から10おきにつけられていく

②行番号を100、増分を20と指定した場合

```
auto 100,20⏎
100*print⏎
120*print⏎
140■
```

行番号が100から20おきにつけられていく

AUTOコマンド

のようにただ⏎キーを叩いた場合は、上に示すように、行番号は10から10おきに、10、20、30のようにつけられていきます。

AUTOコマンドは上に示すような形をしていて、〈行番号〉と〈増分〉を指定することができます。

〈行番号〉のところには、行番号をつけはじめる最初の行番号を指定します。また〈増分〉には、行番号を増やしていく数を指定します。

〈増分〉の指定を省略して、〈行番号〉の指定を50と指定すると、行番号50から10おきに50、60、70と行番号がつけられていきます。

〈行番号〉の指定を100、〈増分〉を20と指定すると、行番号100から20おきに100、120、140と行番号がつけられていくようになります。

AUTOコマンドを解除するには、最後のステートメントを入力して⏎キーを叩くと、カーソルはつぎの行の先頭に移動しますから、そこで[STOP]キーか[CTRL]キーと[C]キーを一緒に押すと、AUTOコマンドは解除されます。

第5章

ディスクの操作

システムディスクのバックアップ

キーボードのキーを叩いてパソコンに入力したプログラムは、ディスクに記録して保管します。パソコンのメモリに記憶されているプログラムをディスクに記録することを、セーブ（SAVE）といいます。

そしてディスクにセーブしたプログラムは、必要としたときに、ディスクからパソコンのメモリに読み込ませて使います。ディスクに記録したプログラムを、パソコンのメモリに読み込ませることを、ロード（LOAD）といいます。

この章では、プログラムをディスクにセーブしたり、ディスクからプログラムをロードする方法について取りあげます。

プログラムをディスクにセーブしたりロードしたりするには、そのために使うシステムディスクのバックアップと、ディスクのフォーマットという準備が必要です。

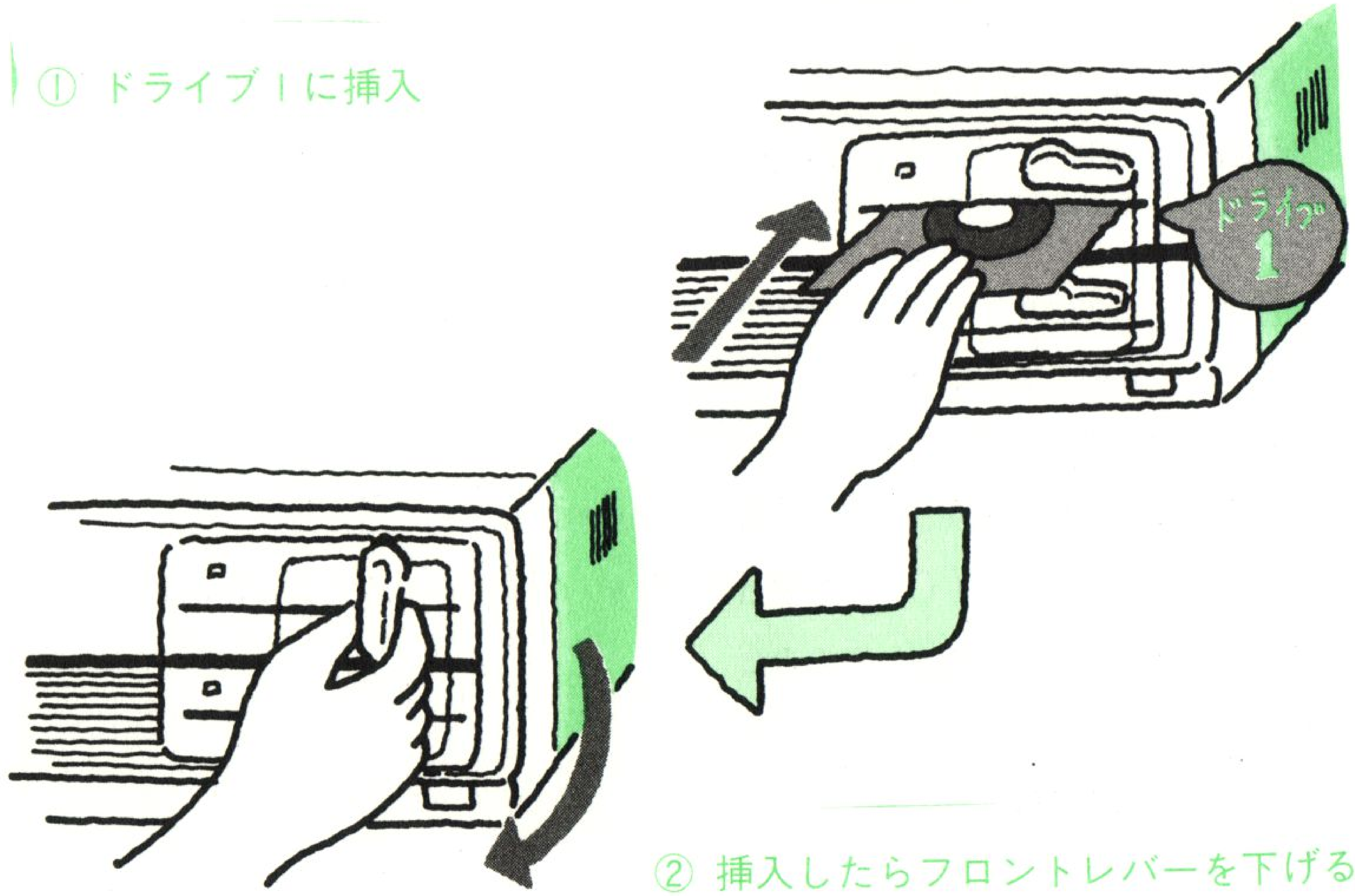
システムディスクのバックアップ

ＰＣ－９８０１ＶＦ２のシステムディスクは、システムディスク（１）とシステムディスク（２）と書かれた二枚のシステムディスクで成り立っています。

ＰＣ－９８０１ＶＭ２のシステムディスクは一枚にまとめられています。

このシステムディスクには、ディスク装置にディスクを挿入して使えるようにするプログラムや日本語を入力することができるプログラムなど、いろいろ大切なプログラムが記録されています。

この大切なプログラムが記録されているシステムディスクをそのまま使ったのでは、システムディスクが壊れたときに取り返しがつきませんから、システムディスクに記録されているものを別のデ



システムディスクをドライブ１に挿入

ィスクにまるごとコピーします。

このように壊れては困るディスクの内容を、別のディスクにまるごとコピーすることを、バックアップを取る、あるいはバックアップコピーを取るといいます。

そして、まるごとコピーしたほうのディスクをシステムディスクとして使い、もとのシステムディスクは、マスターディスクとして大切に保管しておき、システムディスクが壊れたら、ふたたびマスターディスクをまるごとコピーして使います。

ＤＩＳＫベーシックを起動させる

システムディスクのバックアップを取るには、まず、ＤＩＳＫベーシックを起動させます。

ＤＩＳＫベーシックを起動させるには、ＰＣ－９８０１ＶＦ２やＶＭ２に電源を入れ、ディスク装置のドライブ１にシステムディスクを挿入します（上の図参照）。ＶＦ２の場合は、システムディスク（１）をドライブ１に挿入します。

システムディスクをドライブ１に挿入したら、右を向いているフロントレバーを下にさげます。

```
Disk version
How many files(0-15)?■
```
ディスクを扱うことができるベーシックが使えることを示す

```
Disk version
How many files(0-15)?⏎
NEC N-88 BASIC(86) version 3.0
Copyright (C) 1983 by NEC Corporation
192484 Bytes free
Ok
run "menu"⏎
```
⏎キーを叩く

RUN "MENU" と入力して⏎キーを叩く

DISKベーシックの起動とRUN "MENU" の入力

これでＤＩＳＫベーシックが起動して、しばらくすると上に示すように、メッセージが表示されます。

最初に表示されるＤｉｓｋ ｖｅｒｓｉｏｎ（ディスクバージョン）は、ＤＩＳＫベーシックが起動したことを示しています。

このように、ＤＩＳＫベーシックが起動してはじめて、ディスクを操作することができます。

もちろん、Ｄｉｓｋ ｖｅｒｓｉｏｎが表示された状態であっても、Ｄｉｓｋベーシックの起動と同時に、ＲＯＭベーシックも起動するようになっているので、第四章で取りあげたステートメントも使うことができます。

二行目に表示されている、ＨＯＷ ｍａｎｙ ｆｉｌｅｓ（０―15）？は無視してもかまいませんから⏎キーを叩きます。すると、三行のメッセージが表示されます。これで実際にＤＩＳＫベーシックが使える状態になりましたから、

RUN "menu"

と入力します。

【 ユーティリティ ＆ デモンストレーション 】

フロッピィディスクの初期化	(format.nip)
フロッピィディスクのコピー	(backup.n88)
ファイルのコピー	(xfiles.n88)
メモリースィッチの設定	(switch.n88)
ハードディスクの初期化	(format.hd)
文節辞書ファイル・メンテナンス	(dicmen.n88)
利用者定義文字メンテナンス	(mkfont.n88)
プログラム・オートスタートのセット	(setinf.n88)
ＤＩＳＫ ＣＯＤＥのコピー	(sysgen.nip)
ハードディスクの障害復旧	(recov .hd)
ハードディスクのディレクトリ表示	(dir .hd)
ハードディスクのファイル退避／復旧	(backup.hd)
５” １Ｄ／２Ｄコンバーター	(DDconv.n88)
デモンストレーションの起動	(demo)

《 ↑キー、↓キーで選択してＲＥＴＵＲＮキーを押すと起動されます。 》
《 説明が必要な場合は、選択したのちＨＥＬＰキーを押してください。 》

メニューの表示

メニューの表示

RUN"menu"とは、メニューを表示させるコマンドで、入力したあと⏎キーを叩くと、メニューが画面に表示されます（上図参照）。

RUN"menu"の入力を間違えて、Ｓｙｎｔａｘ ｅｒｒｏｒが表示されたら、カーソルキーでカーソルを訂正するところに移動し、間違っているところを訂正します。

表示されたメニューの最初の項目のうえには、横に細長いカーソルが表示されます。このカーソルは、↓（カーソルコントロール）キーを押すと下に移動し、↑キーを押すと上に移動しますから、↓と↑キーでカーソルを移動させて、実行させる項目を選びます。

この場合は、二番目にあるフロッピーディスクのコピー（backup.n88＝バックアップ88）が選ぶ項目ですから、↓キーでカーソルを一つ下に移動します。

このように、選ぶ項目のうえにカーソルを移動させたら、⏎キーを叩きます。

```
Back up a disk

Mount master disk on drive
Master drive#?■
```
←マスターディスクを何番のディスクに挿入したか聞いてくる

```
Back up a disk

Mount master disk on drive
Master drive#? 1⏎
```
←ドライブ1に挿入したので[1]を入力して⏎キーを叩く

ディスクの挿入（1）

すると、ディスクのバックアップを行うbackup.n88というプログラムがスタートしてメッセージが表示されます。

表示されたメッセージのうち、Back up a diskは、「ディスクのバックアップをします」、Mount master disk. on driveは、「マスターディスクをドライブに挿入してください」、Master DRIVE#?は、「マスターディスクを何番のディスクドライブに挿入しましたか」という意味です。

この場合、マスターディスク、つまりバックアップを取るシステムディスクは、ドライブ1に挿入していますから、[1]と入力して⏎キーを叩きます。

新しいディスクをドライブ2に挿入

⏎キーを叩くと、つぎの頁に示すようにメッセージが表示されます。

Mount new disk on driveは、「新しいディスクをドライブに挿入してください」、New drive#?は、「新しいディ

```
Mount master disk on drive

Master drive#? 1⏎


Mount new disk on drive

New drive#?■
```

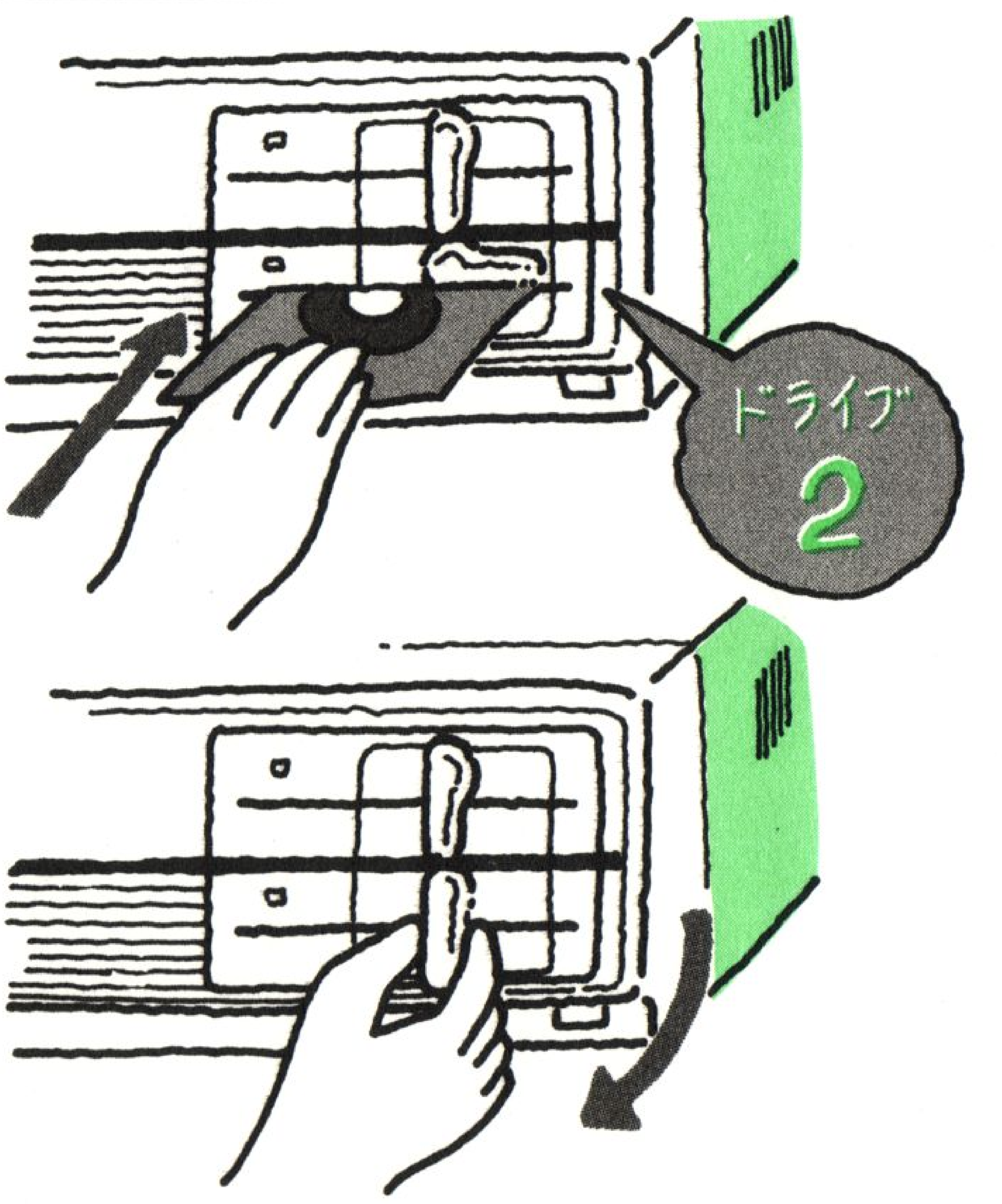
← 新しいディスクを何番のドライブに挿入したか聞いてくる

```
Mount new disk on drive

New drive#? 2⏎
```

← 新しいディスクをドライブ2に挿入して、2と入力して⏎キーを叩く

ディスクの挿入（2）



新しいディスクをドライブ2に挿入

スクを何番のドライブに挿入しましたか」という意味です。

この場合、バックアップを取るシステムディスクはドライブ1に挿入していますから、新しいディスクはドライブ2に挿入して、フロントレバーを下にさげます。

そして、2と入力して⏎キーを叩きます。

```
New drive#? 2⏎

Back up a disk

Do you need physical formatting(y/n)?■
```

バックアップを行うというメッセージ（Back up a disk）

ディスクのフォーマットを行うかどうか聞いてくる（Do you need physical formatting(y/n)?）

```
Back up a disk

Do you need physical formatting(y/n)? y⏎
```

ディスクのフォーマットを行わないと、バックアップを取ることができないのでYを入力して⏎キーを叩く

ディスクのフォーマット（1）

すると、上に示すようにメッセージが表示されます。

■ディスクのフォーマット

Back up a diskは、「ディスクのバックアップを行います」、Do you need physical formatting（y／n）？は、「ディスクのフォーマットを行いますか」という意味です。

新しいディスクは、フォーマットを行わないと使うことができませんから、Yと入力して⏎キーを叩きます。

ディスクのフォーマットとは、その機種のディスク装置でディスクを使うことができるように、ディスクに電気的な処理をほどこすことです。

たとえばPC－9801VM2の場合は、VM2のディスク装置でディスクが使えるように、ディスクに電気的な処理をほどこします。

このように新しいディスクは、それぞれの機種に応じたフォーマットを行うことで、はじめて、その機種で使うことができるようになっています。

```
Back up a disk

Do you need physical formatting(y/n)? y⏎
Physical formatting a disk on drive  2
Sure(y/n)?■
```

←メッセージを表示して確認してくる

```
Physical formatting a disk on drive  2
Sure(y/n)? y⏎
Physical formatting a disk on drive  2
```

← Yを入力して⏎キーを叩く

↑メッセージを表示してディスクのフォーマットを開始

ディスクのフォーマット(2)

さて、フォーマットを行うYと入力して⏎キーを叩くと、上に示すように、Sure(y/n)?というメッセージを表示して、フォーマットされていない新しいディスクが確かにドライブ2に挿入されているかどうか、確認を求めてきます。

フォーマットを行うディスクはドライブ2に挿入されていますから、Sure(y/n)?に対して「y」と入力します。

すると、Physical formatting a disk on drive 2というメッセージを表示して、ドライブ2に挿入したディスクのフォーマットを開始します。

■バックアップの開始と終了

しばらくして、ディスクのフォーマットが終了すると、つづけてディスクのバックアップが開始されて、ドライブ1に挿入したシステムディスクに記録されている内容が、ドライブ2に挿入したフォーマットされた新しいディスクにコピーされていきます。そして、コピーされるたびにトラックナンバーが表示されていきます。

```
Physical formatting a disk on drive  2

Copying track 0
Copying track 1
Copying track 2

Copying track 150
Copying track 151

Completed.
Ok
```

フォーマットを開始して終了するとつづけてバックアップを行う

コピーをするたびにトラックナンバーを表示

システムディスク（1）のバックアップの終了

ディスクのバックアップと終了

システムディスクに記録されているものが全部、ディスクにコピーされると、バックアップは終了して、Completed（完了しました）というメッセージが表示されます。

ところでPC－9801VM2の場合、システムディスクは1枚ですから、一度でバックアップは終わりですが、VF2の場合は、システムディスク（1）とシステムディスク（2）の二枚ありますから、システムディスク（2）のバックアップも取らなければなりません。

システムディスク（2）のバックアップを取るには、システムディスク（1）のバックアップを取ったあと、システムディスク（1）をドライブ1に挿入したままで、つぎのように入力して、

load "backup.n88"

⏎キーを叩いて、システムディスク（1）に記録されているbackup.n88を、LOADコマンド、（211頁参照）でメモリにロードします。それから、ドライブ1に挿入されている、システムディスク（1）を取り出して、システムディスク（2）を

システムディスク(1)をドライブ1に挿入し
load "backup.n88でプログラムをメモリにロード

↓

RUNコマンドでプログラムを実行させる

↓

```
Back up a disk
Mount master disk on drive
Master drive#? 1⏎
```

メッセージが表示されたらシステムディスク(1)を抜き出してシステムディスク(2)をドライブ1に挿入する

↓

```
Mount new disk on drive
New drive#? 2⏎
```

システムディスク(2)のバックアップを取る新しいディスクをドライブ2に挿入する

システムディスク(2)のバックアップを取る手順

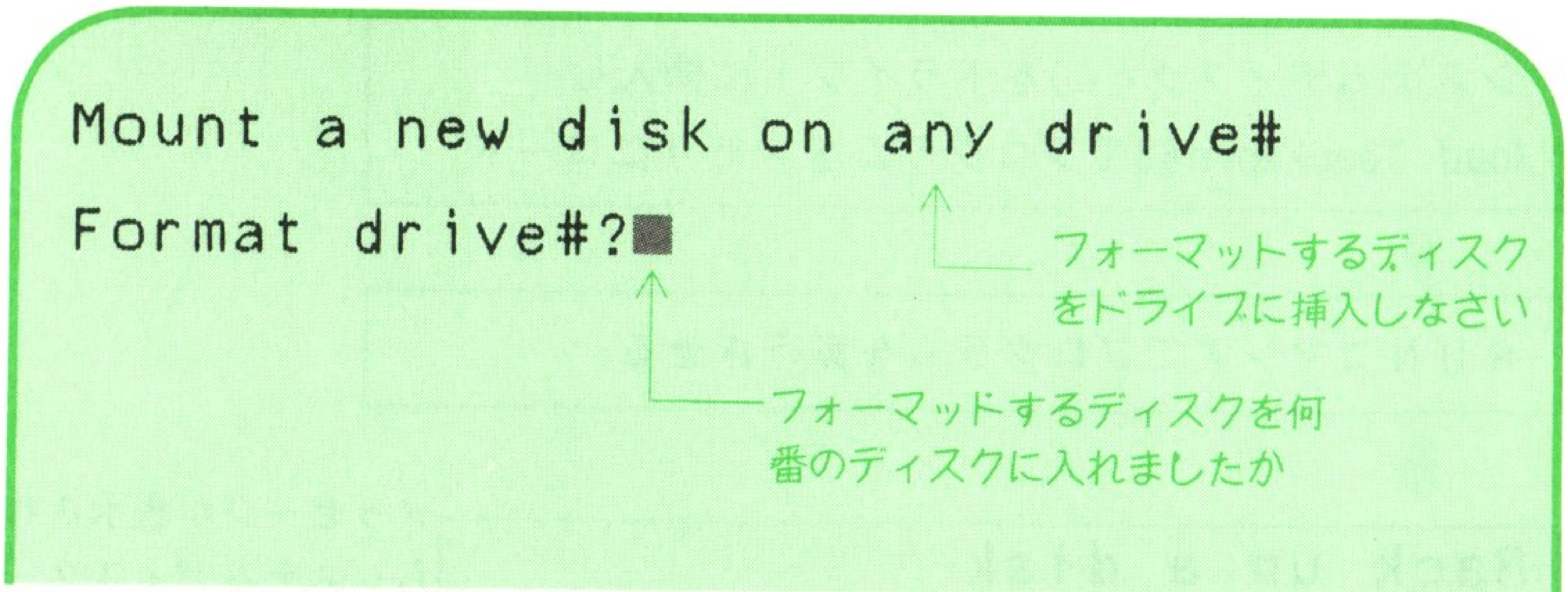
ドライブ1に挿入し、[f・5]キーを押してRUNコマンドでプログラムを実行させます。

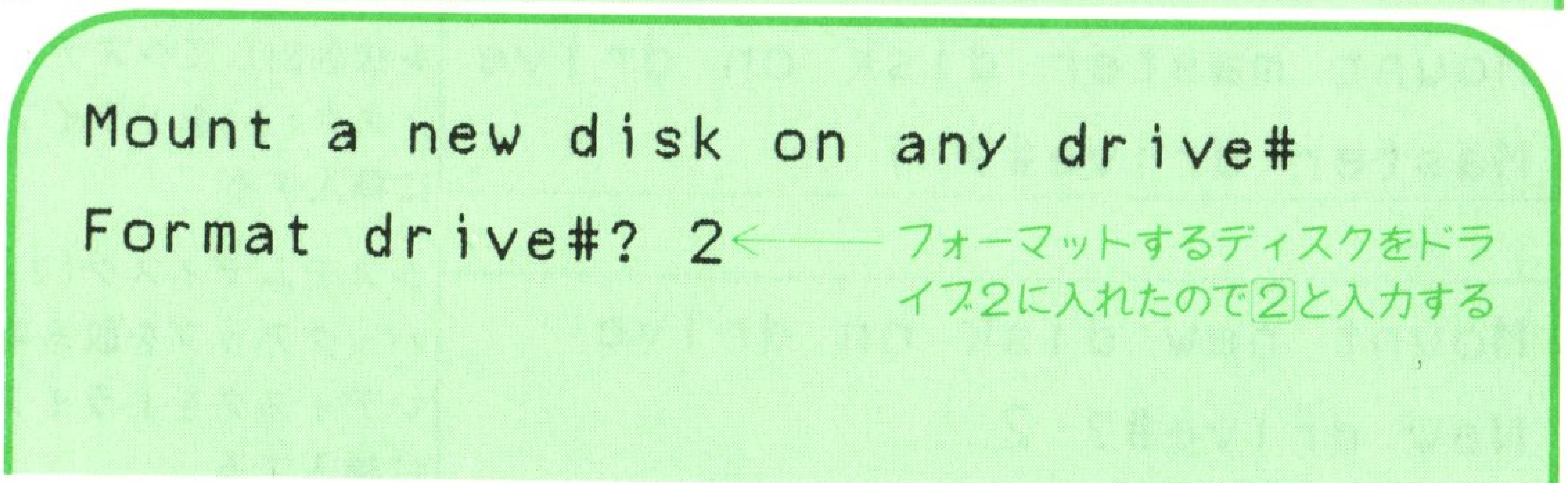
このあとのことは、システムディスク（1）のときとまったくおなじで、上に示すようにメッセージが表示されます。

この場合、バックアップを取るシステムディスクはドライブ1に挿入しましたから、Master drive#?に対して[1]と入力して[⏎]キーを叩きます。

つぎにメッセージが表示されたら、システムディスク（2）のバックアップを取るためのフォーマットのされていない新しいディスクをドライブ（2）に挿入して、New drive#?に対して[2]と入力して[⏎]キーを叩きます。

これで、新しいディスクのフォーマットとシステムディスク（2）のバックアップが開始されます。





フォーマットするディスクの挿入

ディスクのフォーマット

つぎは、ディスクのフォーマットです。

パソコンのメモリに記憶されているプログラムをディスクにセーブするには、198頁で示したように、ディスクのフォーマットを行って、その機種でディスクが使えるように、ディスクに電気的な処理をほどこしておかなければなりません。

なお、ディスクのフォーマットのことを、ディスクの初期化ともいいます。

ディスクのフォーマットを行うには、システムディスクのバックアップのときとおなじように、システムディスクをドライブ1に挿入して、DISKベーシックを起動させておき、run "menu"で、メニューを表示させます。

メニューが表示されると、カーソルはフロッピーディスクの初期化の項目のうえに表示されますから、そのまま⏎キーを叩くと、メッセージが表示されます。

```
Format drive#? 2↵
Do you need physical formatting(y/n)?■
```

2と入力して↵キーを叩く

フォーマットが必要かどうか聞いてくる

```
Format drive#? 2↵
Do you need physical formatting(y/n)? y↵
Formatting a disk 2
```

Yと入力して↵キーを叩く

「ドライブ2に挿入されているディスクをフォーマット中」というメッセージを表示してフォーマットを開始

ディスクのフォーマットを開始

Mount a new disk on drive#は、「新しいディスクをいずれかのドライブに挿入しなさい」という意味です。Format drive#?は、「フォーマットする新しいディスクを何番のドライブに挿入しましたか」という意味です。

ドライブ1にはシステムディスクが挿入されていますから、フォーマットする新しいディスクはドライブ2に挿入することになります。

そこで、新しいディスクをドライブ2に挿入して、Format drive#?に対して[2]と入力して[↵]キーを叩くと、バックアップのときのように、Do you need physical formatting(y/n)?(フォーマットが必要ですか)と聞いてきます。

この場合は、ディスクのフォーマットを行うわけですから、当然「y」と入力して[↵]キーを叩くと、うえに示すようにメッセージを表示して、ドライブ2に挿入されているディスクのフォーマットが開始されます。

```
Formatting a disk 2
Create system disk(y/n)?■
```
↑ディスクのフォーマットが終了するとシステムディスクを作るか聞いてくる

```
Create system disk(y/n)? n↵
Working
Completed.
Ok
■
```
← [N]を入力して[↵]キーを叩く

Completed. ← ディスクのフォーマットが完了する

ディスクのフォーマットの終了

Formatting a disk 2は、「ドライブ2に挿入されているディスクをフォーマット中」という意味です。

しばらくすると、ディスクのフォーマットは終了して、上に示すようにメッセージを表示します。

Create system disk (y/n)?は、「システムディスクを作りますか」という意味ですが、実際には、さきにバックアップを取ったようなシステムディスクを作るのでなく、DISKベーシックを起動させるシステムをコピーするかということです。

ここでは、ディスクのフォーマットだけにしますから、[n]と入力して[↵]キーを叩きます。するとWorkingを表示し、そのあとCompletedを表示します。

Workingは、「フォーマットの残りの作業を行っています」という意味です。Completedは、「ディスクのフォーマットが完了しました」という意味です。

```
Disk version
How many files(0-15)?
NEC N-88 BASIC(86) version 3.0
Copyright (C) 1983 by NEC Corporation
192484 Bytes free
Ok
10 input a
20 b=b+a
30 print b
40 goto 10
```

← ディスクベーシックを起動させる

ディスクベーシックを起動させたらプログラムを入力する

DISKベーシックを起動させたあとプログラムを入力

プログラムのセーブとロード

プログラムをディスクにセーブするには、ROMベーシックの状態ではできず、DISKベーシックの状態で行うようになっています。

したがって、プログラムの入力は、どんな場合でも、まずシステムディスクをドライブ1に挿入して、DISKベーシックを起動させてから、キーボードのキーを叩いて行うようにします。

SAVEはプログラムをセーブ

プログラムをディスクにセーブするにはSAVEコマンドを使います。

SAVEコマンドは、つぎの頁に示すような形をしています。

SAVE 〈ファイルディスクリプタ〉

〈ファイルディスクリプタ〉➡ファイル名を指定する

■ファイル名

ファイル名

〈主ファイル名〉.〈拡張子〉

6文字以内　　3文字以内

KEISAN. BAS ←── 拡張子BASはベーシックであることを示す

PHONE. DAT ←── 拡張子DATはデータであることを示す

keisan
phone
} 拡張子の指定は省略できる

SAVEコマンドとファイル名

ＳＡＶＥコマンドの〈ファイルディスクリプタ〉のところには、ファイル名を指定します。

■ファイル名

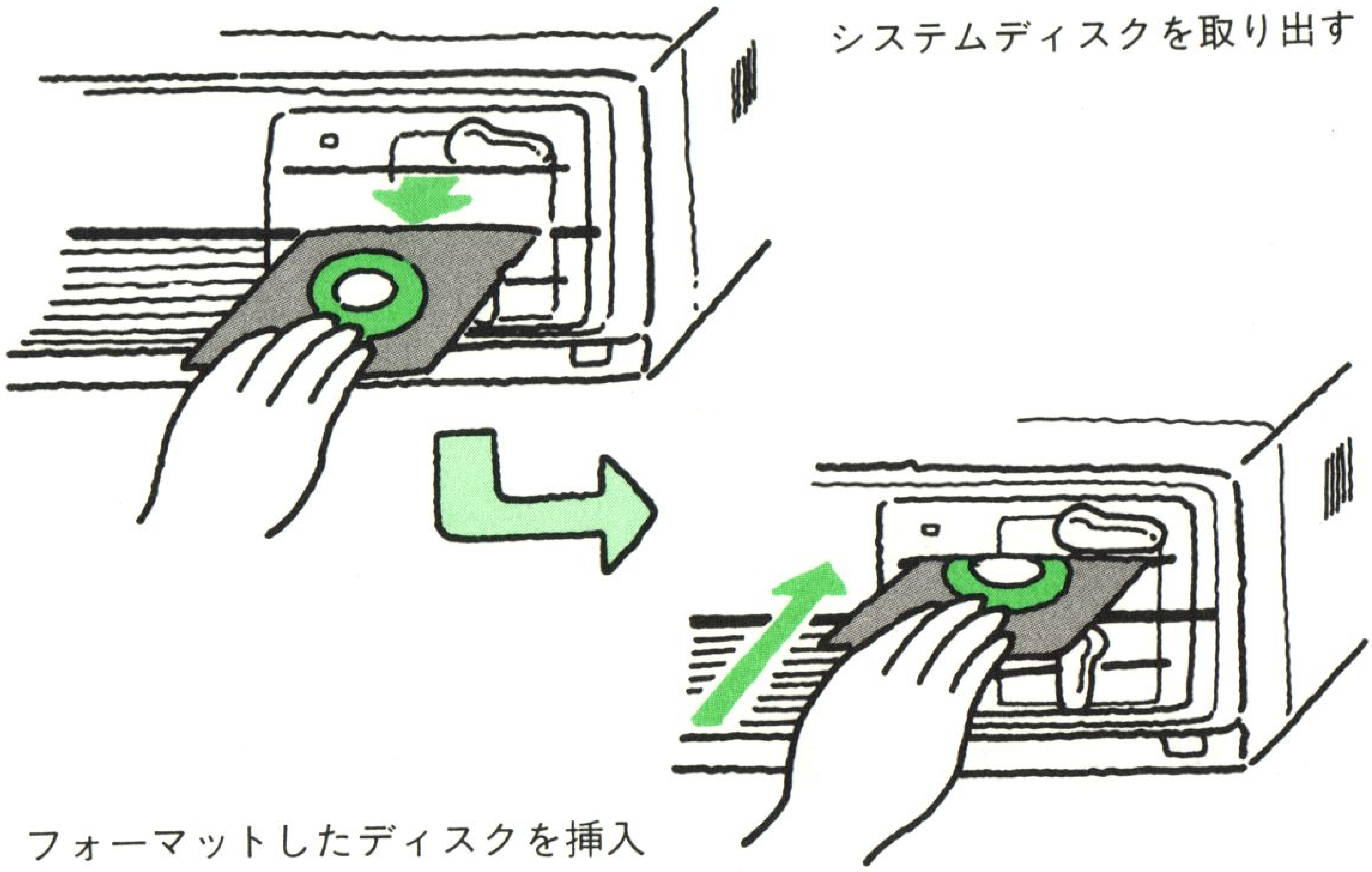
ファイル名とは、ディスクにセーブするプログラムにつける名前で、パソコンはこの名前にしたがって、ディスクに記録されているいろいろなプログラムを識別します。

ファイル名は、主ファイル名と拡張子の二つの部分でできていて、主ファイル名は6文字以内、拡張子は3文字以内でつけることになっています。

ふつう主ファイル名は、ファイルの内容を表わすようにつけ、拡張子はファイルの種類を表わすようにつけます。

たとえば、ディスクにセーブするプログラムがベーシックで書かれた計算のプログラムであれば、主ファイル名をＫＥＩＳＡＮ、あるいはｋｅｉｓａｎとつけます。そして拡張子を、ベーシックを示すＢＡＳ、あるいはｂａｓとつけます。

この場合、主ファイル名だけをＫＥＩＳＡＮとつけて、拡張子をつけなくてもかまいません。



プログラムをセーブするディスクをドライブ1に挿入する場合

ところで、アルファベットには大文字と小文字とがありますが、大文字でつけたファイル名と小文字でつけたファイル名とは、別のファイル名として扱われます。おなじように、ファイル名のなかに一か所でも違うところがあると、別のファイル名として扱われます。したがって、ＫＥＩＳＡＮ、ｋｅｉｓａｎ、Ｋｅｉｓａｎはそれぞれ別のファイル名となります。

ディスクへのセーブ

では実際に205頁で示したプログラムを、ＳＡＶＥコマンドでディスクにセーブしてみます。

プログラムをセーブするディスクは、フォーマットしたディスクですから、このディスクをドライブに挿入しなければなりません。

挿入するドライブはドライブ1、ドライブ2のどちらでもかまいませんが、ドライブ1に挿入するときは、ＤＩＳＫベーシックを起動するために挿入したシステムディスクを取り出してから挿入します。ドライブ2に挿入する場合は、そのまま挿入すればよいわけです。

```
10 INPUT A
20 B=B=A
30 PRINT A
40 GOTO 10
save "■
```
← ファンクションキーの f・6 キーを押して入力

SAVE　ファイル名 ← フォーマットしたディスクがドライブ1に挿入されている場合

SAVE　2：ファイル名 ← フォーマットしたディスクがドライブ2に挿入されている場合

SAVEコマンドの入力とドライブの指定

フォーマットしたディスクをドライブ1、ドライブ2のいずれかに挿入したら、SAVEコマンドを入力します。SAVEコマンドは、ファンクションキーの f・6 キーを押すと、SAVE〟と入力することができます。

SAVEコマンドを入力したら、ファイル名を「〃」引用符で囲んで指定します。この場合、フォーマットしたディスクがドライブ1に挿入されているときは直接ファイル名を指定してもかまいませんが、フォーマットしたディスクがドライブ2に挿入されているときは、ドライブ2を指定する「2：」のあとにファイル名を指定します。

この場合、ファイル名を〝keisan〟としたとすると、

SAVE "keisan"
SAVE "2：keisan"

ということになります。

まず、フォーマットしたディスクがドライブ1に挿入されている場合は、SAVEコマンドにファイル名だけを指定すればよいわけですから、

```
10 INPUT A
20 B=B=A
30 PRINT A
40 GOTO 10
save "keisan"⏎
Ok
```

ファイル名を指定して⏎キーを叩く

セーブを終了

```
10 INPUT A
20 B=B=A
30 PRINT A
40 GOTO 10
save "2:keisan"⏎
Ok
```

ドライブ2のファイル名を指定して⏎キーを叩く

セーブを終了

ディスクへのセーブ

"keisan"と入力して⏎キーを叩きます。すると、ディスクへのセーブが行われてOKを表示します。

ディスクへ正しくセーブできたかどうか確認する方法については、あとで示します。

フォーマットしたディスクがドライブ2に挿入されている場合は、SAVEコマンドのファイル名を指定するところに、まずドライブ2を指定し、つぎにファイル名を指定しますから、"2:keisan"と指定して⏎キーを叩きます。すると、ドライブ2に挿入したディスクへセーブが行われてOKが表示されます。

FILESはファイル名を表示

ディスクにプログラムが間違いなくセーブされたかどうかの確認です。

確認するには、FILESコマンドを使います。

FILESコマンドは、ディスクに記録されているプログラムのファイル名を表示させるコマン

FILES 〈ドライブ番号〉

〈ドライブ番号〉＝ディスクが挿入されているドライブを指定。
ドライブ1の場合は指定を省略できる

```
files⏎      ←ドライブ1の指定を省略して⏎キーを叩く
keisan.     1←ドライブ1のディスクに記録されている
              ファイル名を表示
Ok
```

```
files 2⏎    ←ドライブ2を指定して⏎キーを叩く
keisan.     1←ドライブ2のディスクに記録されている
              ファイル名を表示
Ok
```

FILESコマンド

ドで、上に示すような形をしています。

FILESコマンドの〈ドライブ番号〉のところには、ファイル名を表示させるディスクが挿入されているドライブ番号を指定します。

ドライブ1に挿入されているディスクのファイル名を表示させる場合は、ドライブ番号の指定を省略することができます。

したがって、ドライブ番号の指定を省略して、FILESと入力して⏎キーを叩くと、ドライブ1のディスクに記録されているファイル名が表示されます。

この場合は、ドライブ1のディスクに記録されているファイル名は〝keisan〟だけなので、それだけが表示されます。

FILESコマンドに2を指定して⏎キーを叩くと、ドライブ2のディスクに記録されているファイル名が表示されます。この場合もドライブ2のディスクに記録されているファイル名は〝keisan〟だけなので、それが表示されます。

LOAD〈ファイルディスクリプタ〉

〈ファイルディスクリプタ〉

⬇

ロードするプログラムのファイル名を指定する

LOAD "ファイル名"

LOAD "2：ファイル名"

LOADコマンドとファイル名の指定

「LOADはプログラムをロード」

　ディスクにセーブしたプログラムのロードも、プログラムのセーブとおなじで、ＤＩＳＫベーシックの状態で行います。
　プログラムがセーブされているディスクは、ドライブ1、ドライブ2のどちらに挿入してもかまいませんが、ディスクをドライブ1に挿入するときは、ＤＩＳＫベーシックを起動させるためにドライブ1に挿入したシステムディスクを抜き出してから挿入します。ドライブ2に挿入するときは、そのまま挿入すればよいわけです。
　プログラムをロードするには、ＬＯＡＤコマンドを使います。
　ＬＯＡＤコマンドは、上に示すような形をしています。
　ＬＯＡＤコマンドの〈ファイルディスクリプタ〉のところには、ロードするプログラムのファイル名を指定します。

```
Disk version  ←―― DISKベーシックを起動させる
How many files(0-15)?↵
NEC N-88 BASIC(86) version 3.0
Copyright (C) 1983 by NEC Corporation
192484 Bytes free
Ok
load "■  ←―― f・1キーを押してLOADコマンドを入力
```

```
load "keisan"↵  ←―― ファイル名を入力して↵キーを叩き、
                     ドライブ1のディスクのプログラムを
                     ロード
Ok  ←―― ロード終了
■
```

LOADコマンドの入力とプログラムのロード

ファイル名の指定の仕方はSAVEコマンドのときとおなじで、プログラムをセーブしたディスクがドライブ1に挿入されている場合はファイル名だけを指定し、ディスクがドライブ2に挿入されている場合は、ドライブ2を指定したあとに、ファイル名を指定します。

プログラムのロード

では実際に、プログラムをロードしてみることにします。

さきに、"keisan"というファイル名でプログラムをディスクにセーブしましたから、ここではそのプログラムをロードします。

最初は、ディスクをドライブ1に挿入した場合です。

LOADコマンドは、ファンクションキーの[f・1]キーを押すと入力されますから、ファイル名"keisan"を入力して[↵]キーを叩くと、プログラムのロードが行われてOKが表示されます。これでディスクにセーブされていたプログラムがパソコンのメモリに記憶されます。

```
load "2:keisan"↵
Ok
■
```

load "2:keisan"↵ ← ドライブ2のディスクの場合は、ファイル名の前に「2:」と指定して↵キーを叩く

Ok ← ロード終了

```
load "2:keisan"↵
Ok
list↵
10 INPUT A
20 B=B=A
30 PRINT A
40 GOTO 10
Ok
```

list↵ ← ロードしたプログラムの確認は、LISTと入力して↵キーを叩く

10〜40 ← いまロードされたプログラムが表示される

プログラムのロードと確認

では、プログラムをセーブしているディスクがドライブ2に挿入されている場合です。

この場合はさきに示したように、LOADコマンドのファイル名を指定するところに、まずドライブ2を指定し、そのあとにファイル名を指定すればよいわけですから、"2:keisan"と指定して↵キーを叩きます。すると、ドライブ2に挿入されたディスクから、プログラムがロードされてOKが表示されます。

なお、指定したファイル名がディスクにないと、

File not found

というメッセージが表示されます。

■プログラムの確認はLIST

ロードされたプログラムの確認は、LISTコマンドでメモリに記憶された内容を表示させて行います。

LISTはファンクションキーの[f・4]キーを押すと入力されますから、入力したら↵キーを叩きます。すると、いまディスクからロードしたプログラムが画面に表示されます。

KILL〈ファイルディスクリプタ〉

〈ファイルディスクリプタ〉
↓
削除するファイル名を指定

KILL "ファイル名" ← ドライブ1に挿入されたディスクのプログラムを削除する場合

KILL "2：ファイル名" ← ドライブ2に挿入されたディスクのプログラムを削除する場合

KILLコマンドと削除するファイル名の指定

KILLはファイルを削除

ディスクにセーブしたプログラムのうち、不必要なプログラムを削除する場合は、KILL（キル）コマンドを使います。KILLコマンドは上に示すような形をしています。

KILLコマンドの〈ファイルディスクリプタ〉のところには、削除するプログラムのファイル名を指定します。ファイル名の指定の仕方は、SAVEコマンドの場合とおなじで、ドライブ1に挿入されているディスクのプログラムを削除するには、ファイル名だけを指定します。ドライブ2に挿入されているディスクのプログラムを削除するには、ドライブ2とファイル名を指定します。

たとえば、つぎのように入力して⏎キーを叩くと、指定された"keisan"というファイル名のプログラムが削除されます。

KILL "keisan"
KILL "2：keisan"

第6章

日本語の入力

```
コノ プ°ログ"ラム ハ ゴ"ウケイ ヲ
モトメル プ°ログ"ラム デ"ス。

タス カズ" ヲ ニュウリョク シテクダ"サイ?■
```

```
このプログラムは合計を
求めるプログラムです。

足す数を入力してください?■
```

漢字・かな混りの表示のほうがわかりやすい

文節変換で日本語を入力する

今日のパソコンは、漢字・かな混りの日本語を入力して使うことができます。日本語を使うことができるようになったことが、パソコンを非常に使いやすいものにしています。

たとえば、ＰＲＩＮＴ文でガイド文を表示するとき、上に示すようにカタカナで表示すると、非常にわかりにくいものになりますが、漢字・かな混りの文で表示すると、わかりやすくなります。

日本語を入力するには辞書が必要です。

ＰＣ－９８０１ＶＦ２、ＶＭ２には、日本語を入力する辞書としてＢＵＮＳＥＴ・ＳＵというファイル名の辞書と、ＫＡＮＪＩ・ＤＩＣというファイル名の辞書の二つが、システムディスクに記録されています。

```
Disk version
How many files(0-15)?
NEC N-88 BASIC(86) version 3.0
Copyright (C) 1983 by NEC Corporation / Microsoft
163812 Bytes free
Ok
■
```

DISKベーシックを起動させる

⏎キーを叩く

CTRLキーとXFERキーを押して日本語入力モードにする

R〔かな〕が表示されて日本語入力モードになる

R〔かな〕

CTRL＋XFERキーで日本語入力モードにする

BUNSET・SUは、文節変換で日本語を入力する辞書、KANJI・DICは漢字を一字ずつ入力していって、日本語を入力する辞書です。

ここではBUNSET・SUを使って、文節変換で日本語を入力する方法を示します。

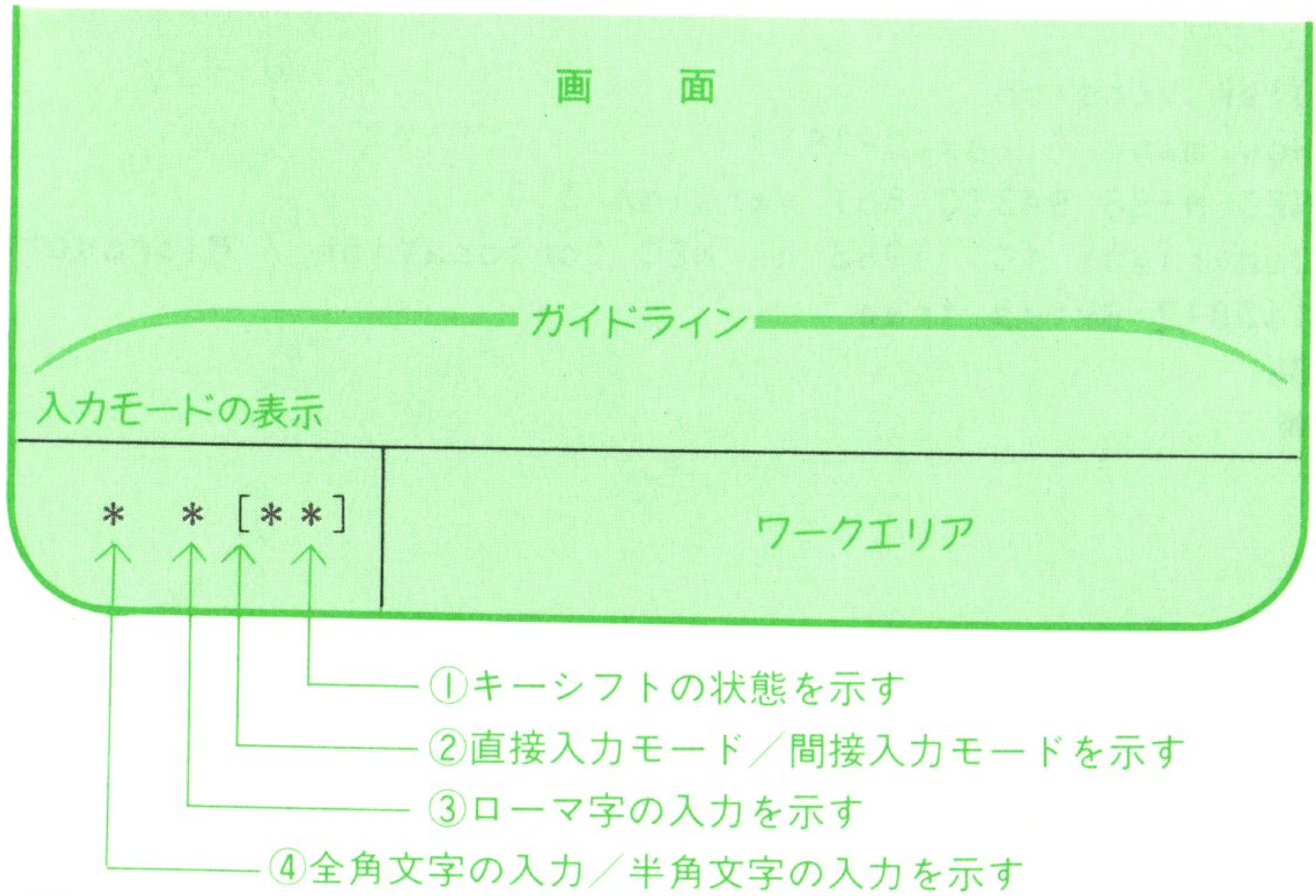
日本語入力モード

日本語を入力するには、PC−9801VM2あるいはVF2に電源を入れ、システムディスクをドライブ1に挿入します。VF2の場合、BUNSET・SUはシステムディスク（2）に記録されているので、システムディスク（2）をドライブ1に挿入します。

DISKベーシックが起動したら、⏎キーを叩いてメッセージを表示させます。そして、CTRLキーを押しながらXFERキーを押すと、

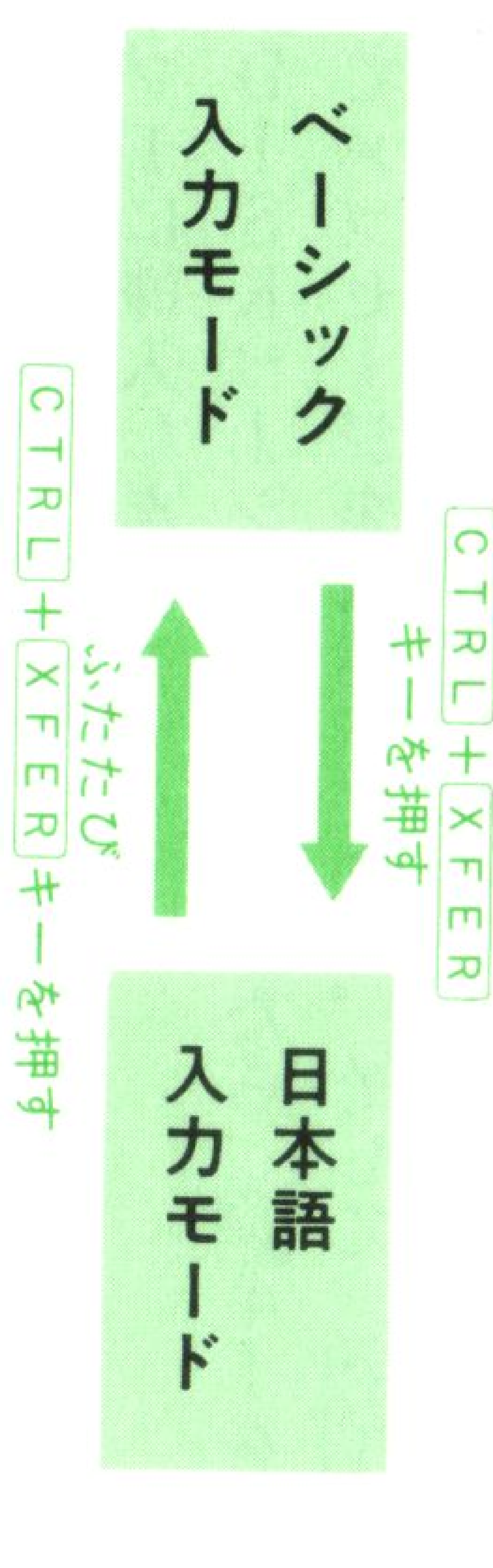
CTRL＋XFER＝日本語入力モードにする

画面の下にR［かな］と表示されて、日本語入力モードになります。



入力モードの表示

日本語入力モードをもとのベーシック入力モードに戻すには、ふたたびCTRLキーを押しながらXFERキーを押せばよいのです。



■入力モードの表示

画面の下のR［かな］が表示されている行を、ガイドラインといいます。また、R［かな］の表示を、上に示すように入力モードの表示といい、その右側をワークエリアといいます。

日本語を入力するには、入力モードの表示が、［かな］あるいは【かな】の状態でしか入力することができませんが、入力モードの表示にはいろいろあるので、つぎに簡単に示しておきます。

① キーシフトの状態を示す

キーシフトの状態を示すところには、かな、カナ、英数などの文字が表示されます。

かな＝ひらがなが入力できる
カナ＝カタカナが入力できる
英数＝英文字、数字、記号が入力できる

② **直接入力モード／間接入力モードを示す**

直接入力モードは細いカギカッコ（「」）、間接入力モードのときは太いカギカッコ（【】）が表示されます。

直接入力モード＝入力した文字は、画面に表示されているカーソルの位置に直接表示される。

この場合、白地リバース（白地に黒）で表示されます。⏎キーを叩くと入力されて、ふつうの表示になります。

間接入力モード＝入力した文字がワークエリアに入力され、⏎キーを叩くと入力されて、画面に表示されているカーソルの位置に表示される。

③ **ローマ字の入力を示す**

ローマ字の入力のときは、R文字が表示され、ローマ字の入力でないときは、R文字が表示されません。

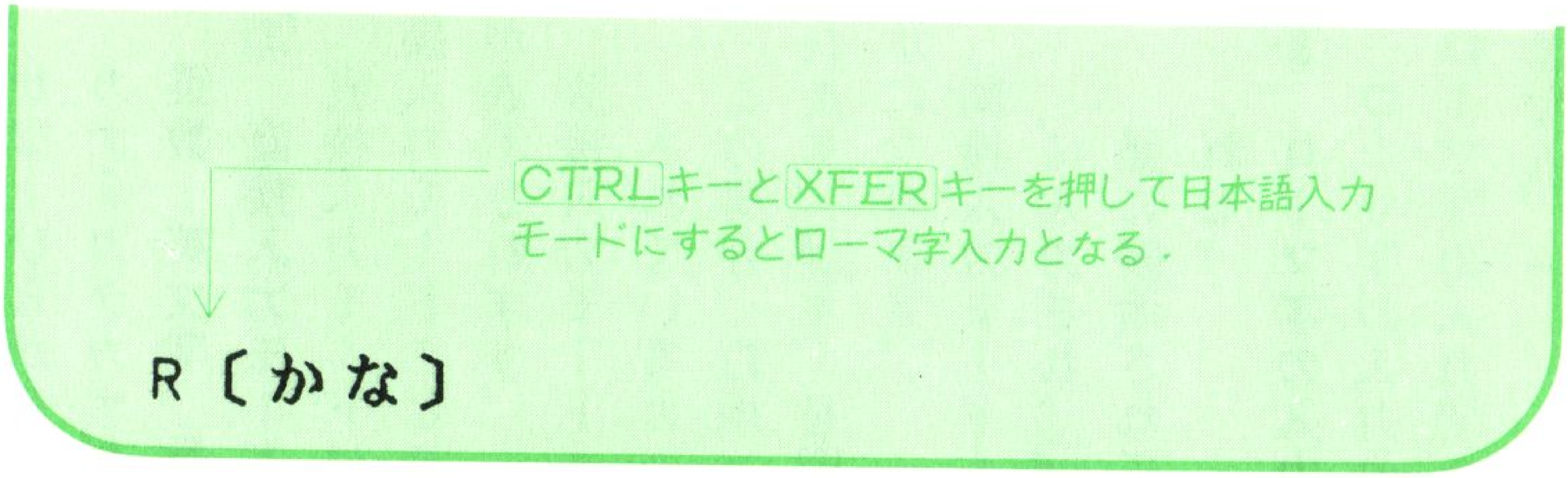
R［かな］＝ローマ字を入力すると、入力したローマ字がひらがなで入力される
R［カナ］＝ローマ字を入力すると、入力したローマ字がカタカナで入力される

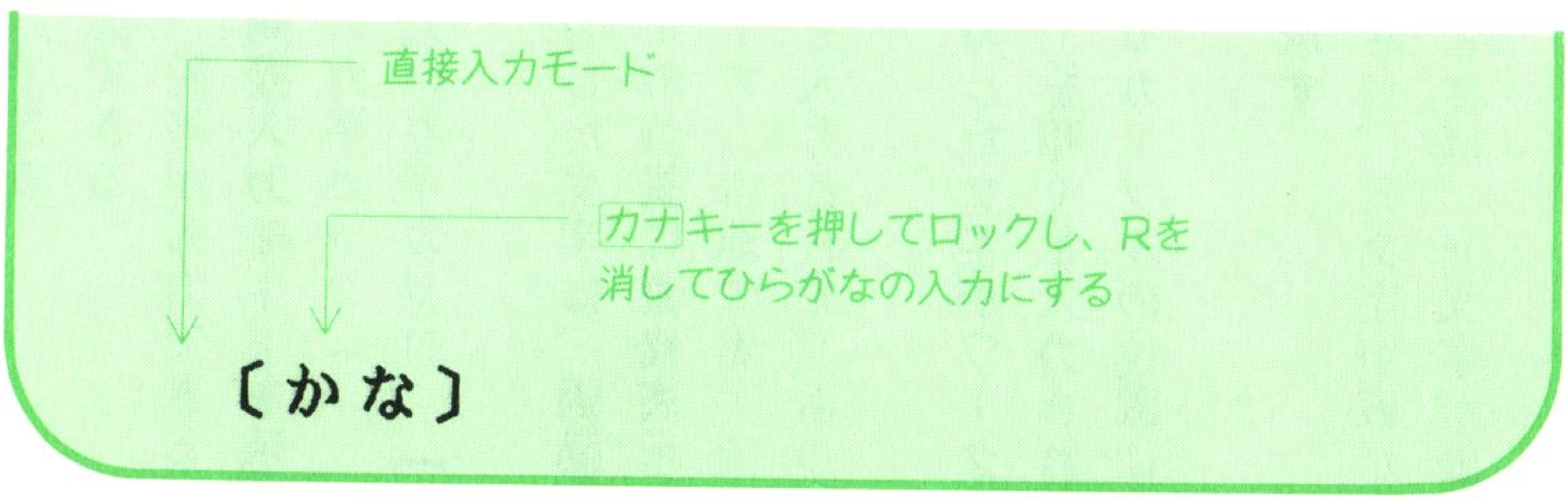
④ **全角文字の入力／半角文字の入力を示す**

半角文字の入力のときは、＊記号が表示され、全角文字のときは、＊記号は表示されません。

＊［英数］＝入力した英文字、数字、記号などが半角文字で表示される
＊［カナ］＝入力したカタカナが、半角文字で入力される







カナキーを押して入力モードの表示を［かな］に

文節とは

日本語入力モードにしたら、漢字・かな混りの日本語の入力です。文節変換で入力します。

文節とは、改めて取りあげるまでもないことですが、／で区切った文の一つ一つをいいます。

今日は／天気だったので、／ゴルフにいった。

つまり、文節変換で日本語を入力するということは、文節ごとにひらがなを入力し、それを漢字に変換するキーを押して、つぎのように、漢字・かな混りの文に変換して入力するということです。

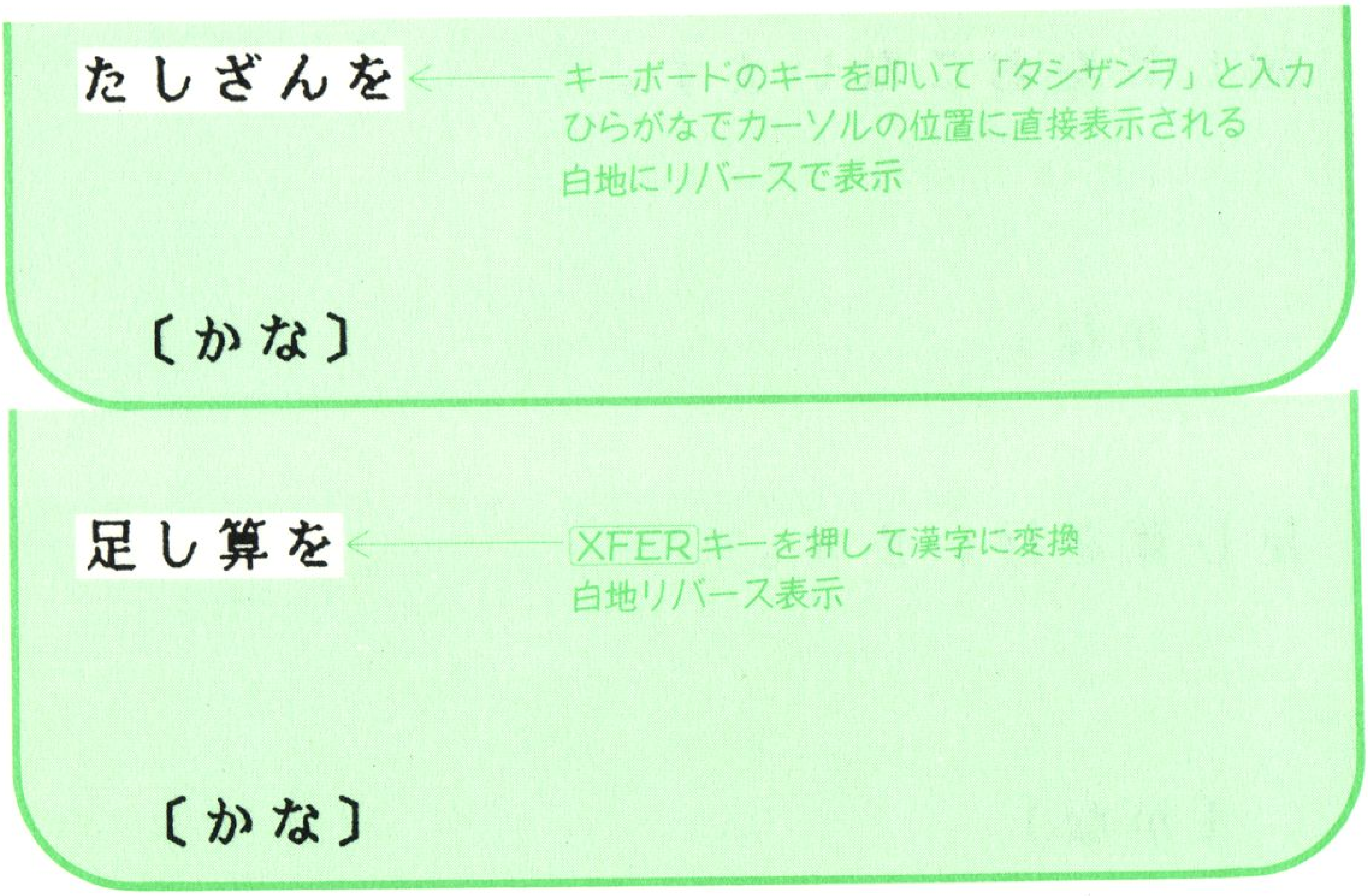
きょうは → 今日は

カナキーを押して［かな］に

文節でひらがなを入力して、漢字に変換するには、入力モードの表示を［かな］にします。

そうするには、カナキーを押してロックすると、うえに示すようにRが消えて、入力モードの表示は［かな］になります。

この場合、かなを囲んでいるカギカッコは細いので直接入力モードですから、入力したひらがなはカーソルの位置に直接表示されます。



XFERキーを押して漢字に変換（1）

直接入力モードで日本語を入力

ここでは、つぎのように文を入力します。

足し算を行います。足す数を入れてください。

最初に入力するのは「たしざんを」です。このように入力するには、カタカナのキーを「タシザンヲ」と叩くと、カーソルの位置から白地リバースで表示されます。

「ざ」の入力は サ と ゛ のキーを叩き「を」の入力は、SHIFTキーを押しながら ヲ のキーを叩きます。

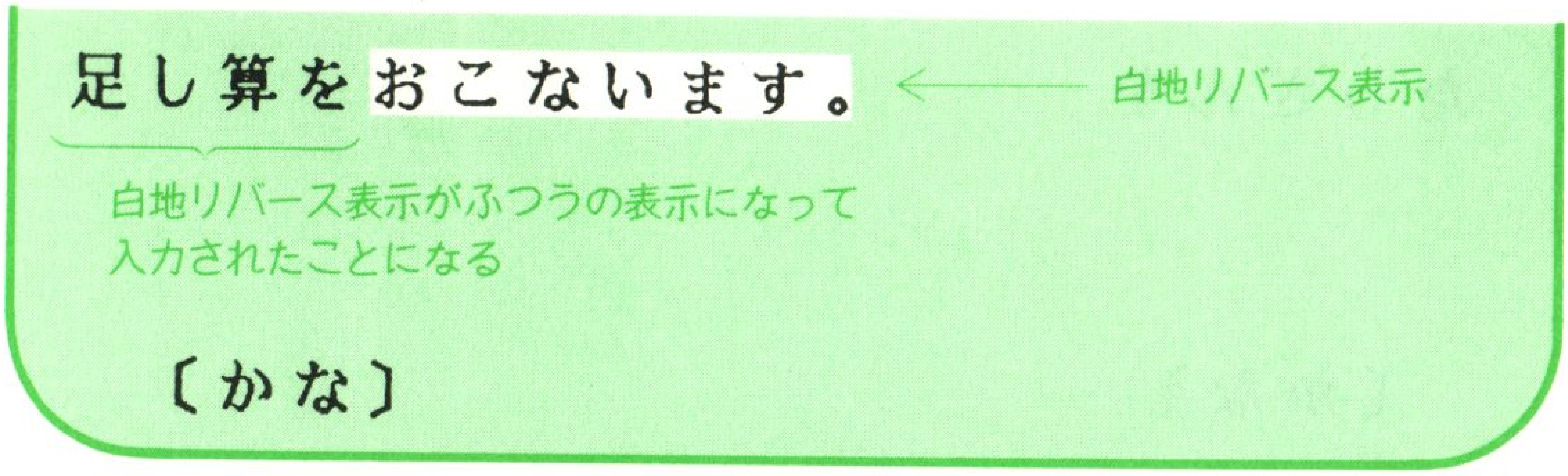
■XFERはひらがなを漢字に変換

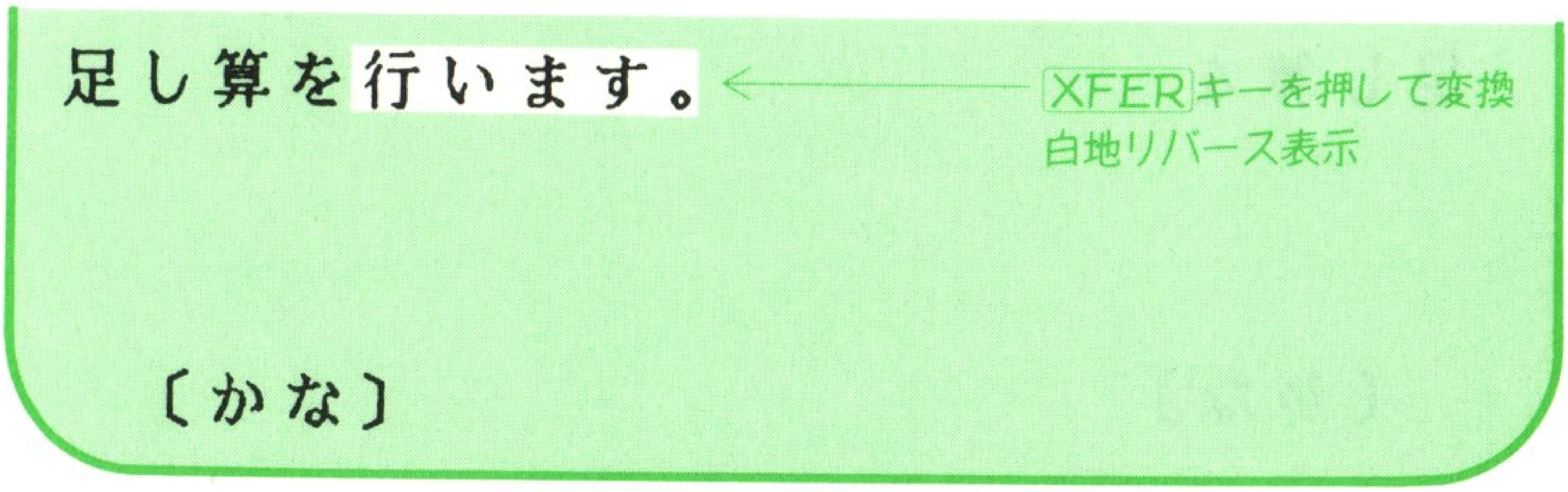
ひらがなを入力したら、XFERキーを押します。XFERキーは漢字に変換するキーです。

XFER＝ひらがなを漢字に変換

すると「たしざんを」が「足し算を」に変換されます。

変換したら、つぎの「おこないます。」を入力します。





[XFER]キーを押して漢字に変換（2）

「。」は[SHIFT]キーを押しながら、[ル]のキーを押すと入力できます。

「おこないます。」を入力すると、カーソルの位置に白地リバースで表示されて、「足し算を」がふつうの表示になります。これで「足し算を」が入力されたことになります。

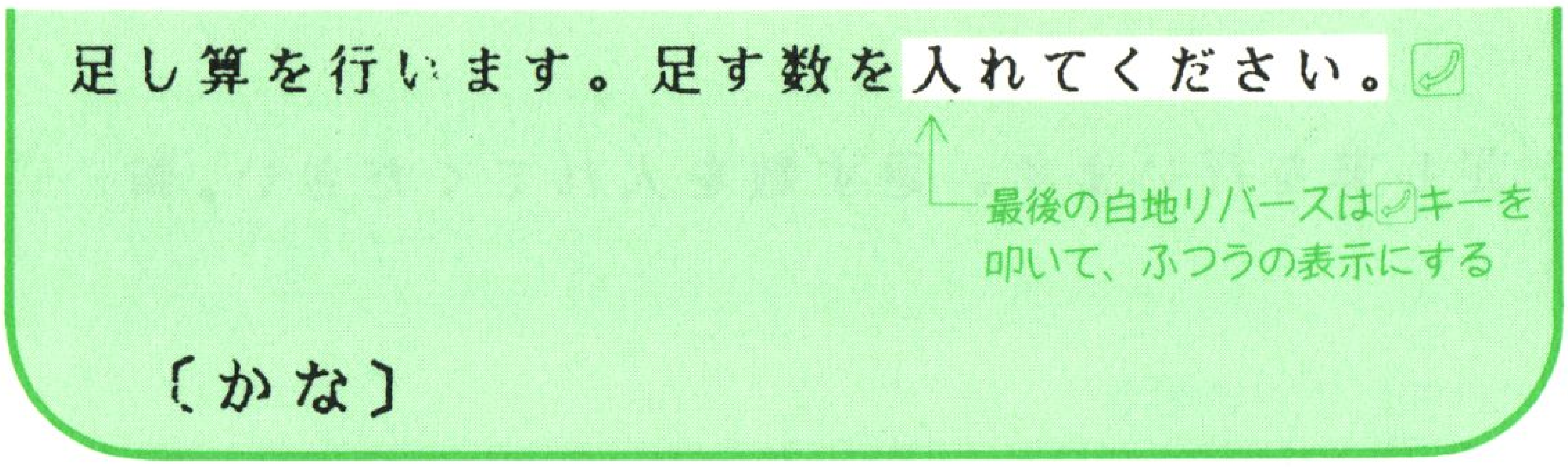
つまり、[XFER]キーを押して漢字に変換しても、白地リバース表示のままではただ変換されただけで、入力したことにはならないということです。つぎにひらがなを入力すると、自動的にまえの白地リバース表示がふつうの表示になって、入力されたことになります。

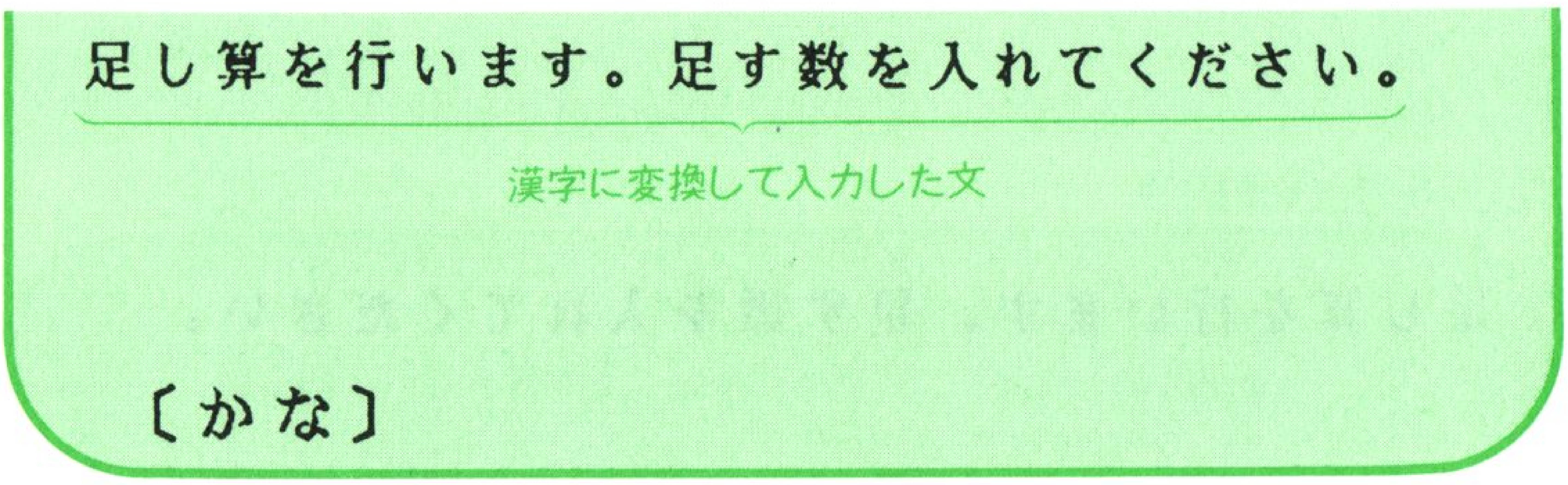
最後の文節を入力する場合は、つぎに入力するひらがながありませんから、[↵]キーを叩きます。[↵]キーを叩くと白地リバース表示は、ふつうの表示になります。

[↵]＝白地リバース表示をふつうの表示にする

さて[XFER]キーを押すと、「おこないます。」が、上に示すように「行います。」に変換されます。

残りの文節も、つぎのようにひらがなを入力し





漢字に変換して入力した文

てXFERキーを押すと、漢字に変換されます。

たす ↓ 足す　かずを ↓ 数を
いれてください。↓ 入れてください。

「入れてください。」は最後ですから⏎キーを叩いて、白地リバース表示をふつうの表示にします。

上に、変換した結果をまとめて示します。

■入力したひらがなの訂正

入力したひらがなを訂正するときには、ESC（エスケープ）キー、BS（バックスペース）キー、あるいはDEL（デリート）キーを使います。

ESCキーを押すと、白地リバース表示の部分がすべて消去されます。

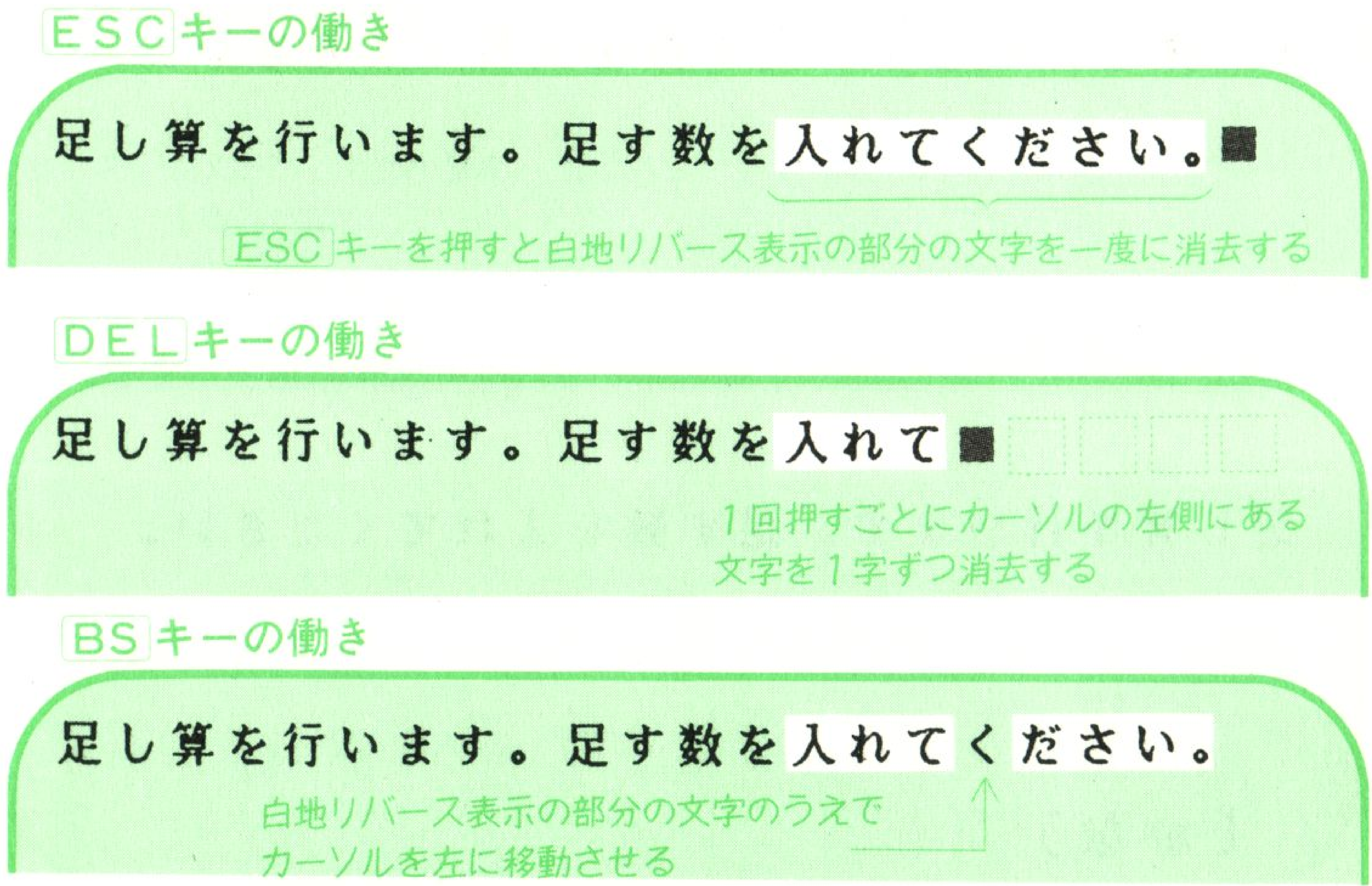
ESC＝白地リバース表示の部分を消去

DELキーの場合は、一回押すたびにカーソルの左側の文字を1字ずつ消去していきます。

DEL＝左側の文字を1文字ずつ消去

DELキーは、白地リバース表示の部分だけでなく、ふつうの表示になった文字も消去します。

BSキーは文字の消去ではなく、一回押すたびに白地リバース表示の部分の文字のうえを、1文



ESC、DEL、BSキーの働き

字分ずつカーソルを左に移動させます。

BS＝カーソルを1字分ずつ左に移動

したがって、BSキーは入力したひらがなを訂正するときに使います。

つまり、BSキーでカーソルを左に移動して訂正するひらがなのうえにもってきたら、カタカナのキーを叩いて、正しいひらがなを入力するというようにします。カーソルを右に移動させるときは、➡キーを使います。

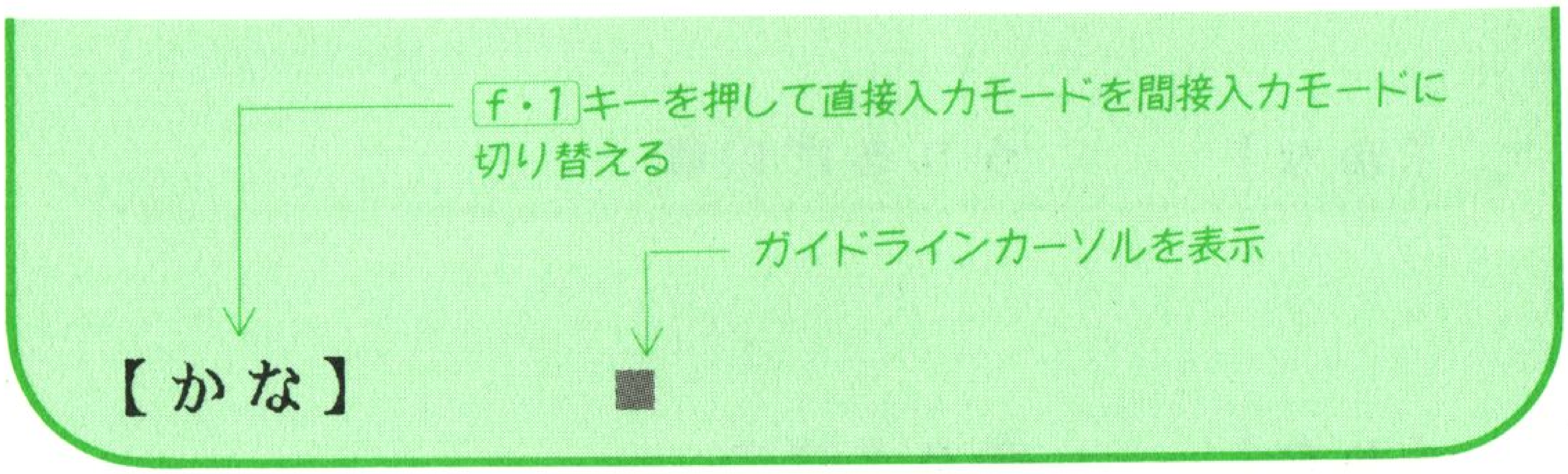
間接入力モードで日本語を入力

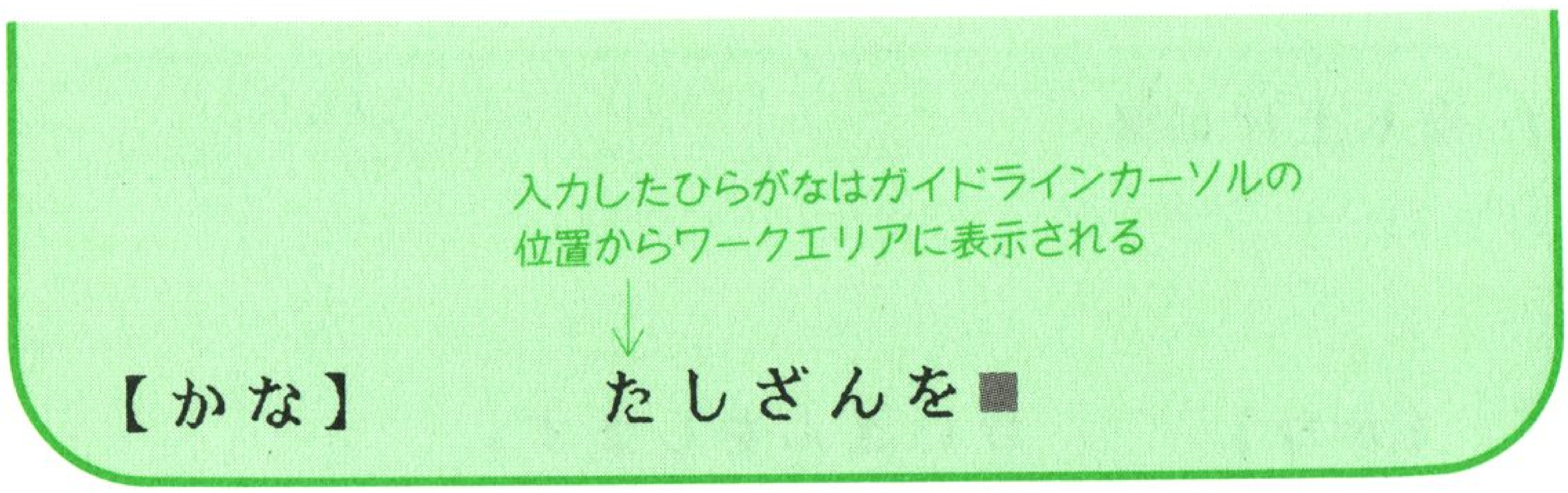
間接入力モードで日本語を入力するには、入力モードの表示を【かな】にします。

入力モードの表示を【かな】にするには、直接入力モードの［かな］が表示されている状態のときに、ファンクションキーのf・1キーを押すと、間接入力モードの【かな】の表示になります。

F・1キーで入力モードを切り替え

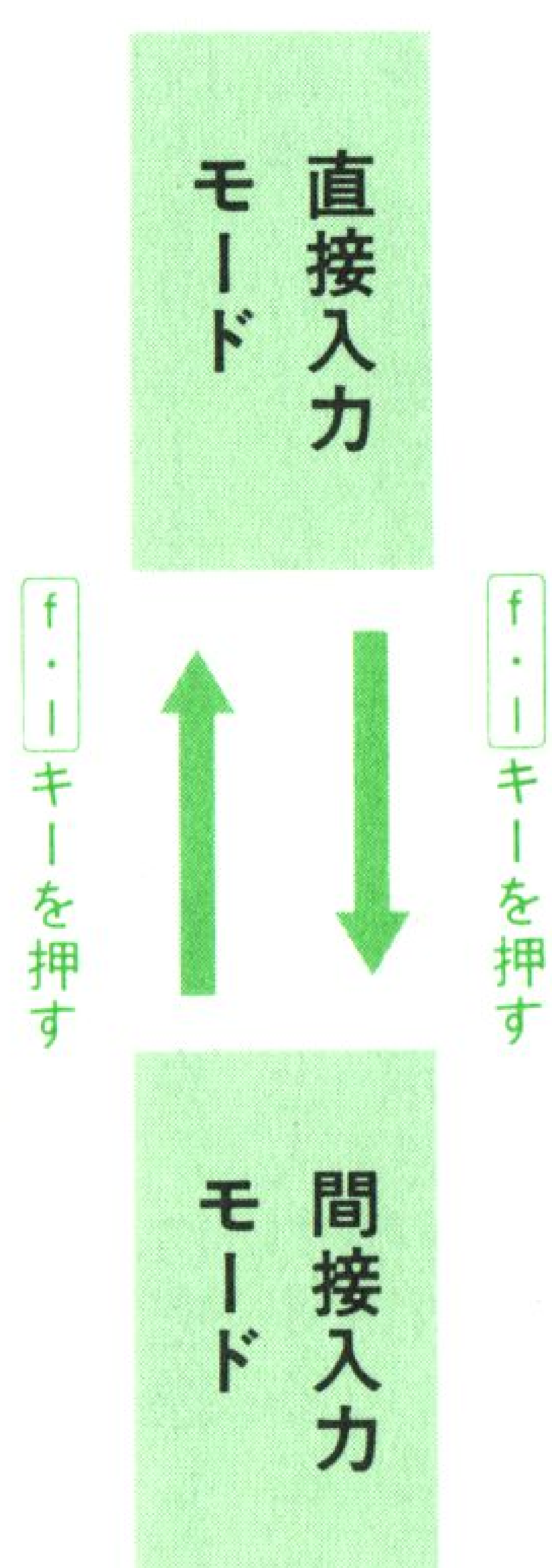
f・1キーは、直接入力モードと間接入力モー





間接入力モード

ドを切り替えるキーで、間接入力モードのとき[f・1]キーを押すと、直接入力モードになります。



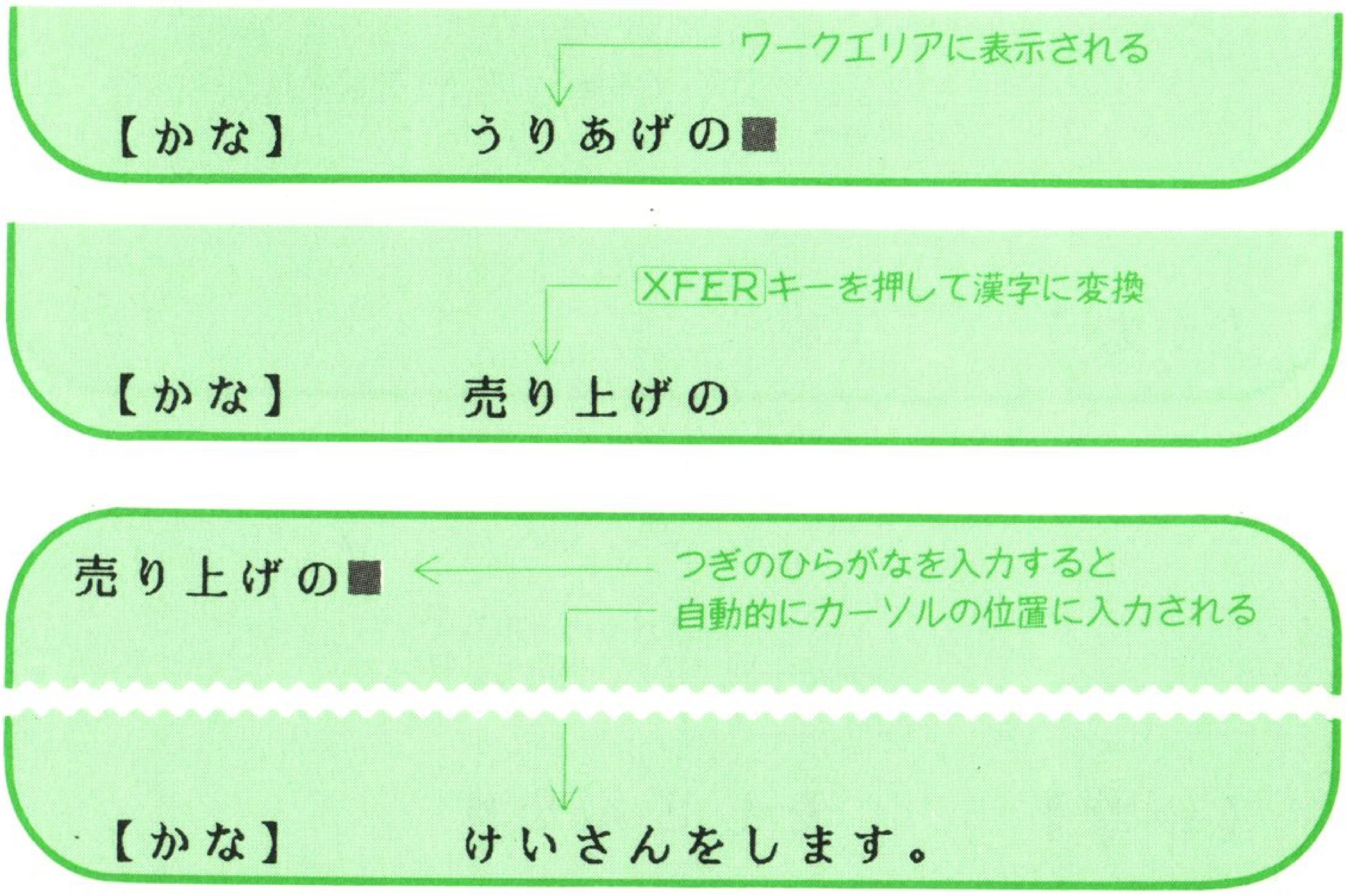
ひらがなはワークエリアに表示

間接入力モードの**【かな】**に変わると、ガイドラインカーソルが表示されます。ガイドラインカーソルが表示されている位置から右が、218頁に示したワークエリアです。

間接入力モードの場合、キーボードのキーを叩いて入力したひらがなは、このガイドラインカーソルの位置からワークエリアに表示されます。

したがって、間接入力モードの場合、ワークエリアに表示されたひらがなを、[XFER]キーを押して、変換することになります。

ここでは、つぎの文を変換して入力することにします。



間接入力モードの漢字への変換と入力（1）

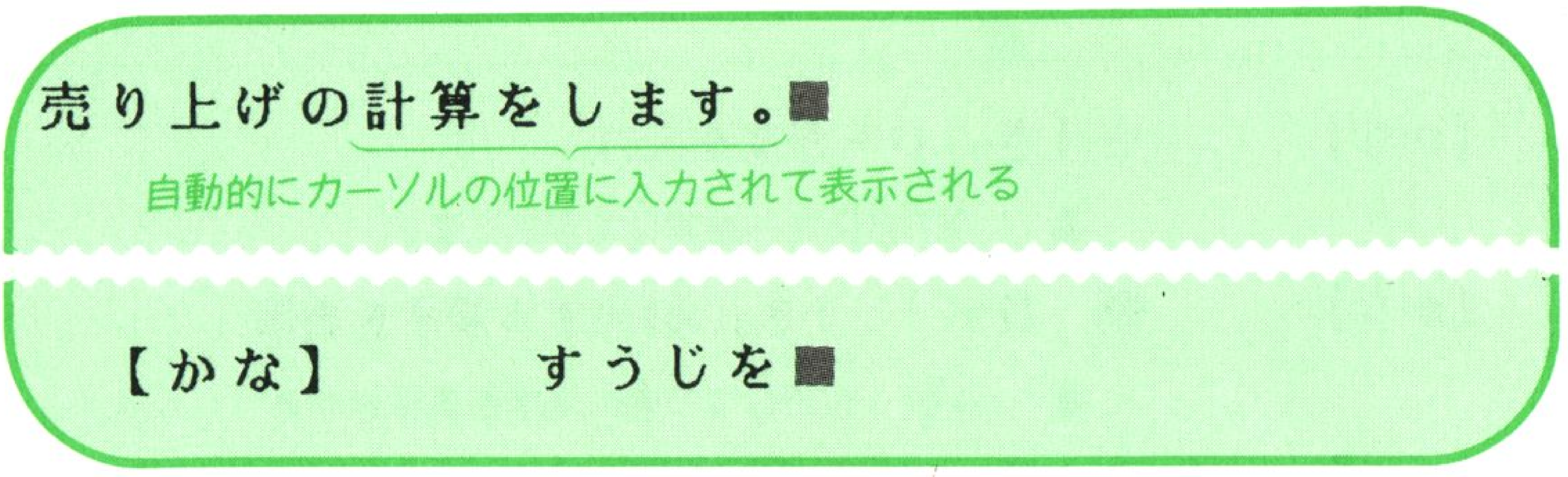
売り上げの計算をします。数字を入力してください

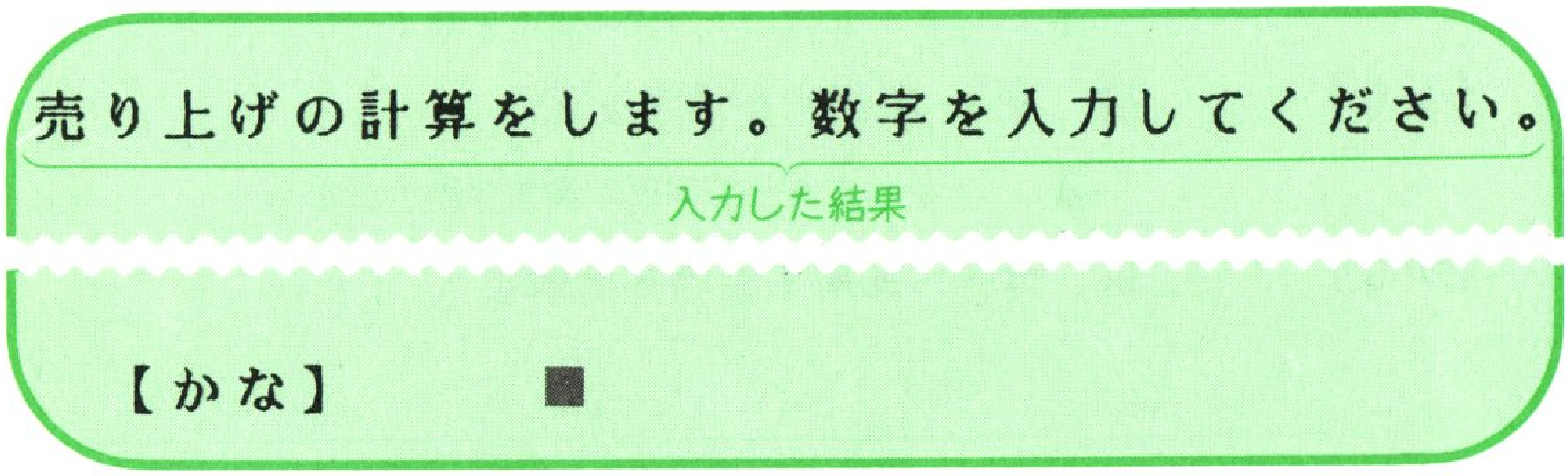
まず「うりあげの」と入力すると、ワークエリアに「うりあげの」と表示されますから、XFERキーを押して「売り上げの」に変換します。つぎに「けいさんをします。」と入力すると、「売り上げの」は自動的に入力されてカーソルの位置に表示され、「けいさんをします。」がワークエリアに表示されます。

XFERキーを押すと「計算をします。」に変換されますから、つぎに「すうじを」と入力すると、「計算をします。」が自動的にカーソルの位置、つまり「売り上げの」のあとに入力されて表示されます。

XFERキーを押して「数字を」に変換したら「にゅうりょくしてください。」と入力すると、「数字を」は自動的にカーソルが表示されている位置に入力されて表示されます。

XFERキーを押して「入力してください。」に変換したら、そのあと入力するひらがながありま





間接入力モードの漢字への変換と入力（2）

せんから、[↵]キーを叩いて入力します。

上に入力した結果を示します。

間接入力モードが直接入力モードと異なる点は、漢字に変換するひらがながいったんワークエリアに表示されるということだけで、漢字に変換する方法などはおなじです。

入力したひらがなの訂正に使う[ESC]キー、[DEL]キー、[BS]キーの働きそのものは直接入力モードの場合とおなじです。

[ESC]キーを押すとワークエリアに表示されている文字が、一度に消去されます。

[DEL]キーを押すと、ガイドラインカーソルの左側に表示されている文字が消去されます。

[BS]キーを押すと、ワークエリアに表示されている文字のうえを、ガイドラインカーソルが左に移動します。

【かな】　とうき←ひらがなを入力

⬇　XFERキーを押して漢字に変換

【かな】　騰　貴←「とうき」に該当する漢字を表示

⬇　⬇キーを押してつぎの漢字を表示させる

【かな】　冬　季←該当する漢字を表示

⬇　⬇キーを押してつぎの漢字を表示させる

【かな】　投　棄←該当する漢字を表示

⬇　⬇キーを押してつぎの漢字を表示させる

【かな】　投　機←変換する漢字を表示

⬇キーで漢字を表示させて選択

漢字への変換と選択

ひらがなを入力してXFERキーを押すと、即座に変換する漢字が表示されましたが、どんな場合でも即座に変換する漢字が表示されるわけではありません。

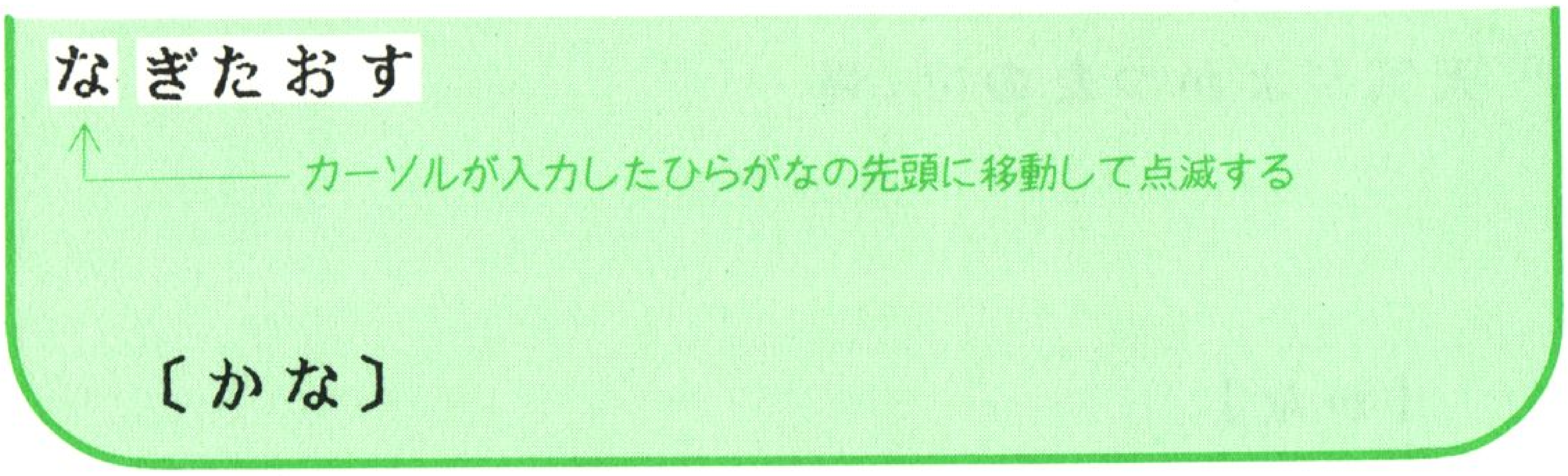
XFERキーを押して、変換する漢字が表示されないときは、⬇キーを押して辞書のなかに登録されている漢字を表示させていき、変換する漢字を選択することになります。

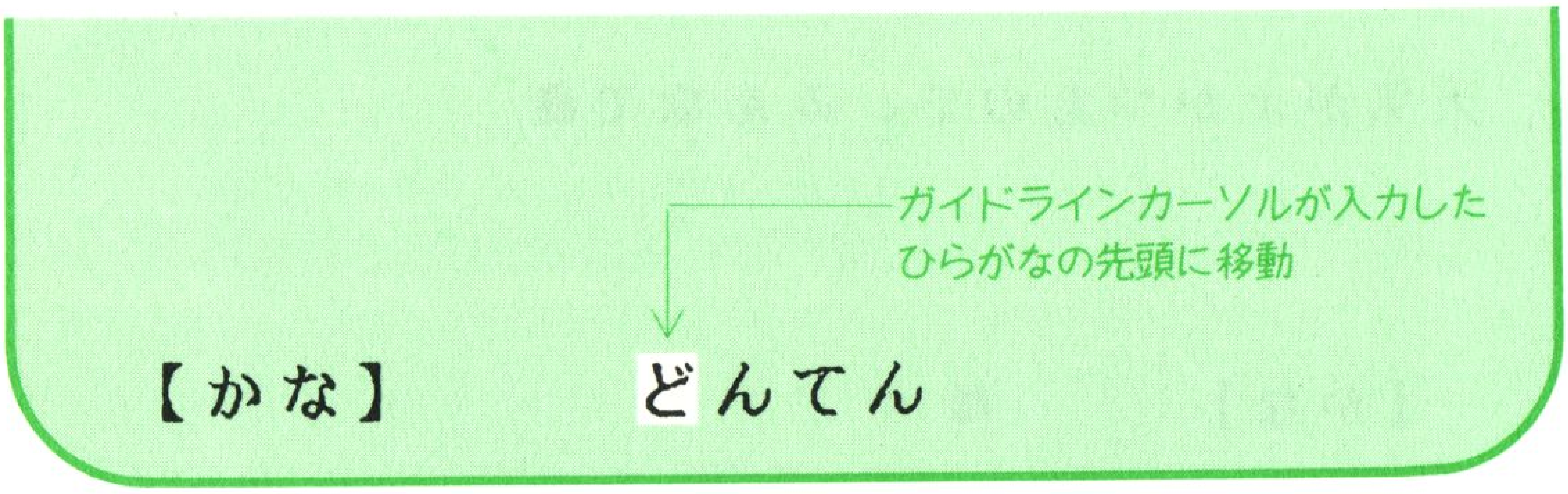
⬇キー＝辞書に登録されている漢字を表示

たとえば、「投機」と漢字を入力するために、「とうき」と入力してXFERキーを押すと、「騰貴」に変換されて表示されますから、⬇キーを押して辞書に登録されている漢字をつぎつぎに表示させて、変換する漢字を選択します。

一度まえに表示された漢字に戻すときは、⬆キーを使います。

⬆キー＝まえに表示させた漢字に戻す





辞書に該当する漢字が一つも登録されていないとき

漢字が登録されていないとき

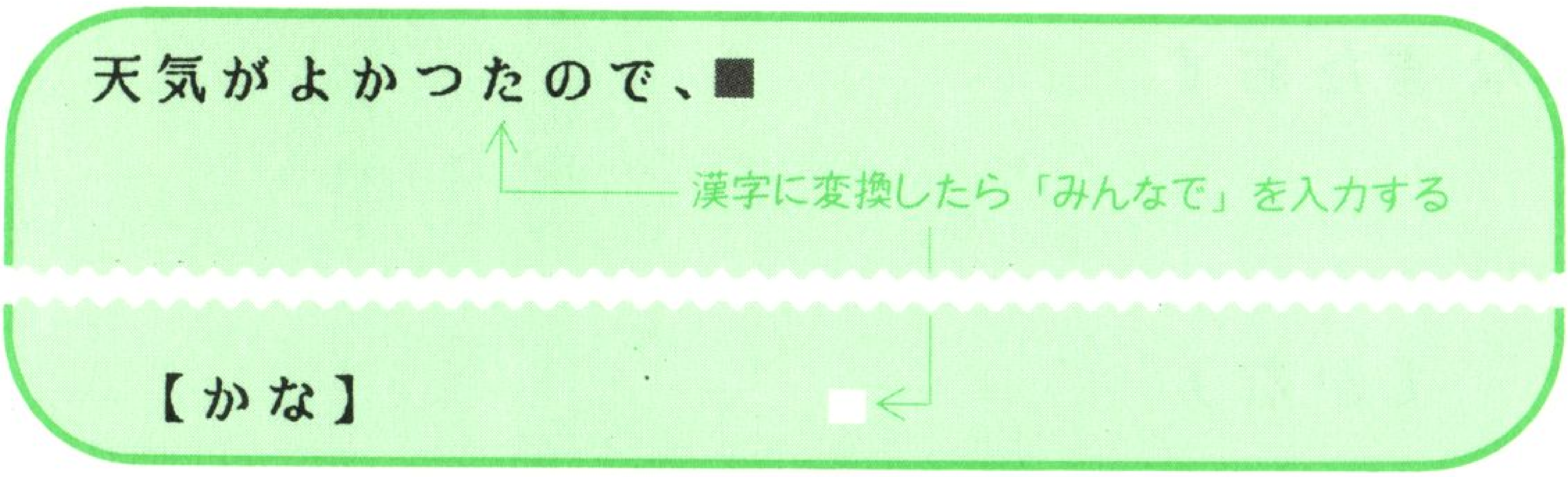
ところで「投機」の場合は、変換する漢字が辞書に登録されていましたが、変換する漢字が辞書に必ず登録されているとはかぎりません。変換する漢字が辞書に登録されていない場合は、⬇キーを押しつづけていくと、入力したひらがなの表示に戻ります。

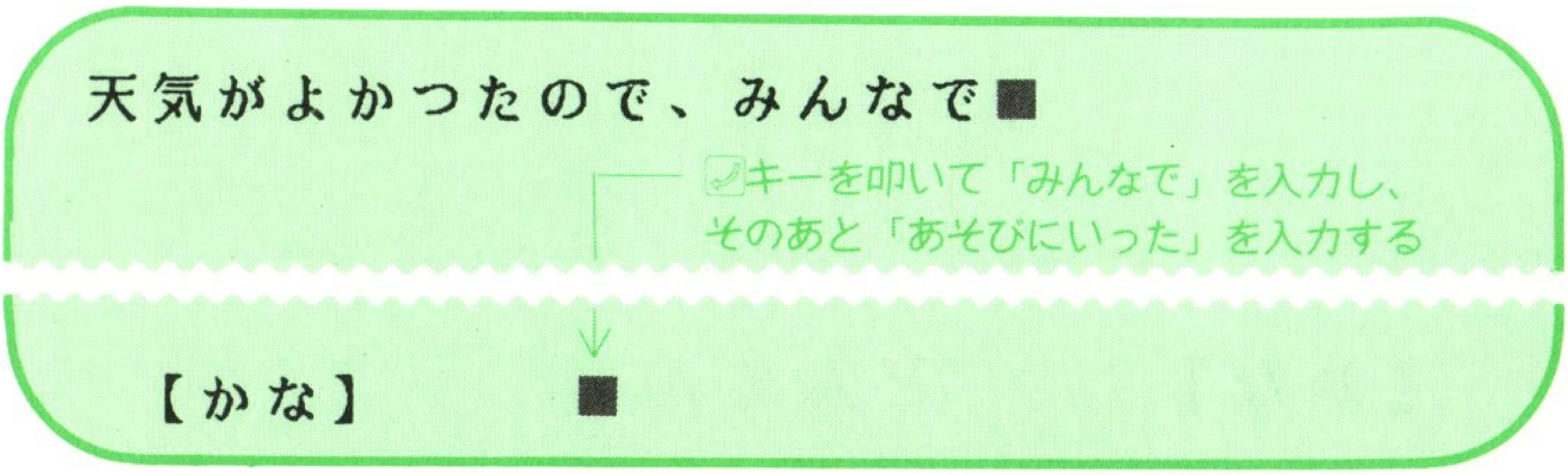
また、入力したひらがなに該当した漢字が辞書に一つも登録されていない場合は、XFERキーを押すと同時に、カーソルあるいはガイドラインカーソルが入力したひらがなの先頭に移動します。

ひらがなの入力(無変換入力)

これまで、漢字に変換するひらがなの入力について示しましたが、漢字に変換しないひらがなの入力もあります。漢字に変換しないひらがなの入力を、無変換入力といいます。

無変換でひらがなを入力する場合には、無変換で入力するひらがなだけを入力し、入力し終わっ





無変換入力は、その部分だけを独自に入力

たら⏎キーを叩きます。

たとえば、つぎの文を間接入力モードで入力するとした場合、

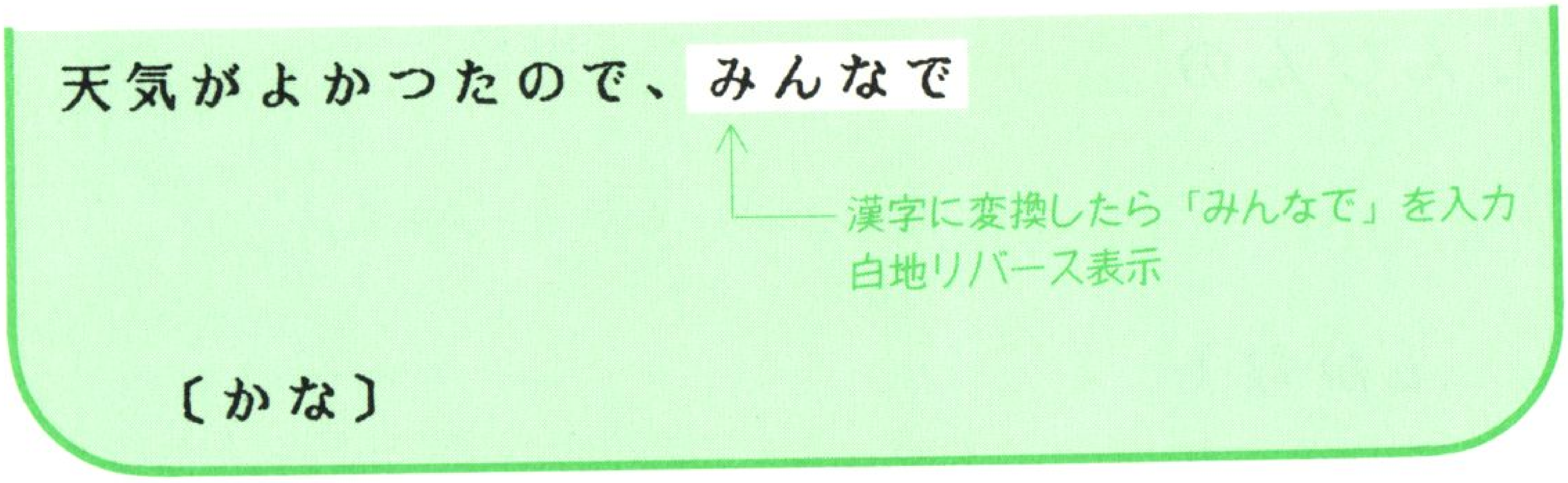
天気がよかったので、みんなで遊びにいった

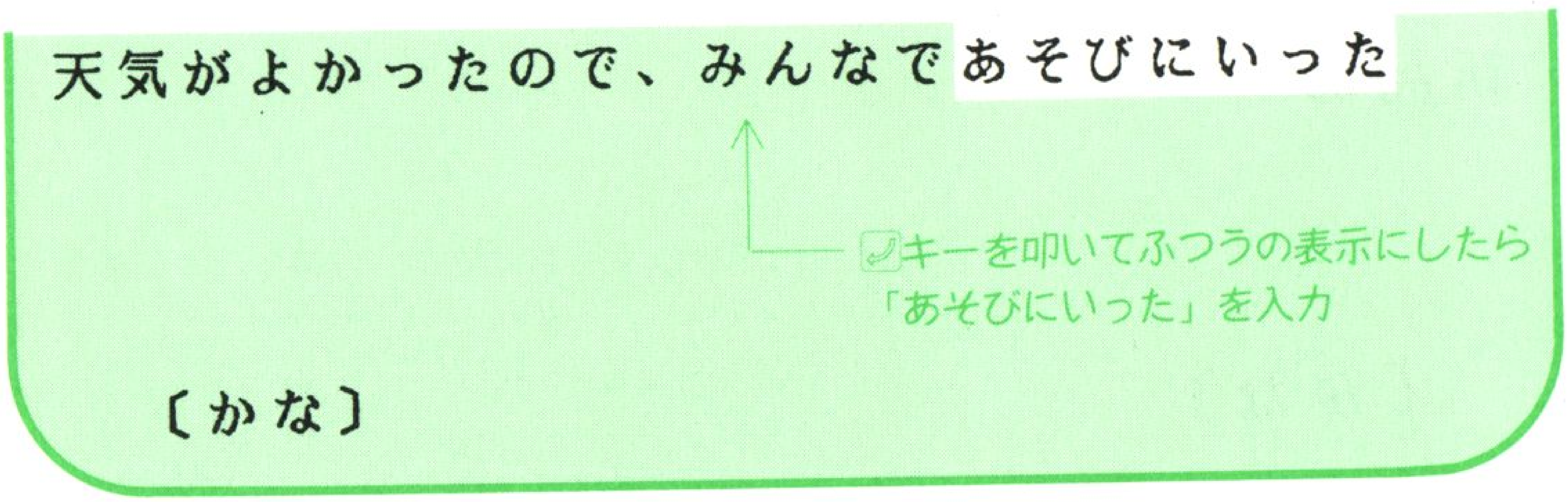
のうち「みんなで」を無変換入力にします。

なぜなら、「てんきがよかったので、みんなで」までを入力すると、[XFER]キーを押しても漢字に変換されないからです。また「みんなであそびにいった」までを入力して[XFER]キーを押すと、「皆であそびにいった」と変換されてしまうからです。

したがって、「てんきがよかったので、」を入力して[XFER]キーを押して漢字に変換したら、「みんなで」を入力し、入力したら⏎キーを叩いてカーソルの位置に表示させます。そのあと、「あそびにいった」を入力して漢字に変換します。

この場合は間接入力モードで示しましたが、直接入力モードの場合もおなじで、「みんなで」を入力したら⏎キーを叩いて白地リバース表示をふつうの表示にし、そのあと「あそびにいった」を入





直接入力モードの無変換入力

力します。

カタカナの入力

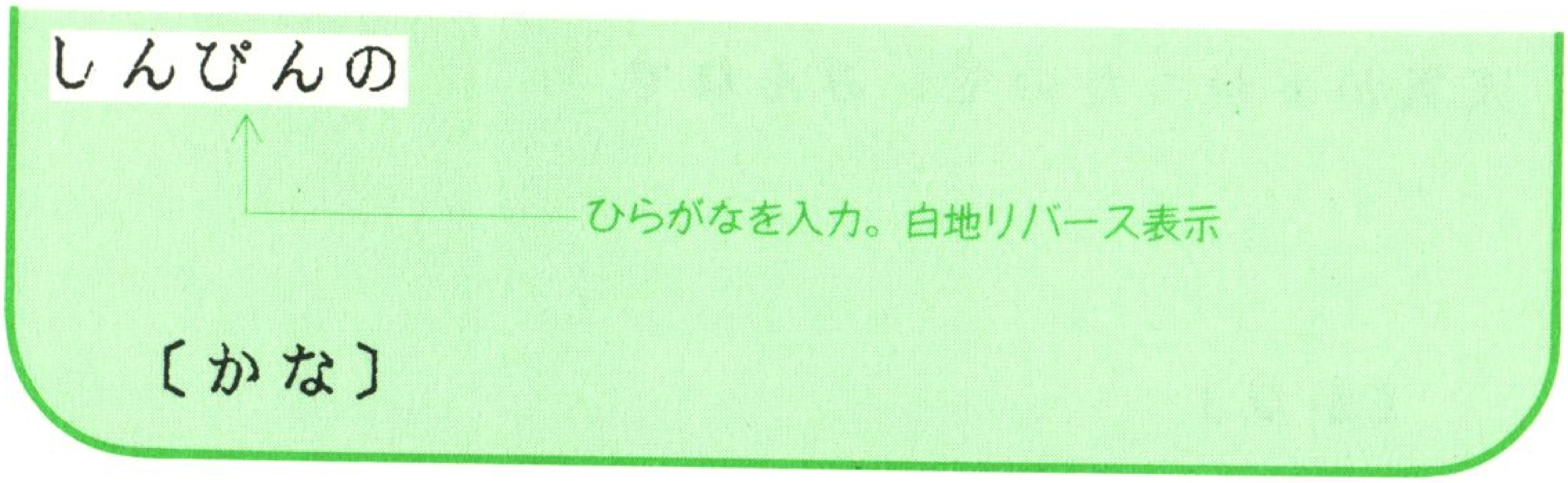
まえの項で日本語入力モードで、漢字とひらがなを入力しましたが、日本語入力モードでカタカナを入力することもあります。そこでここでは、カタカナの入力について示します。

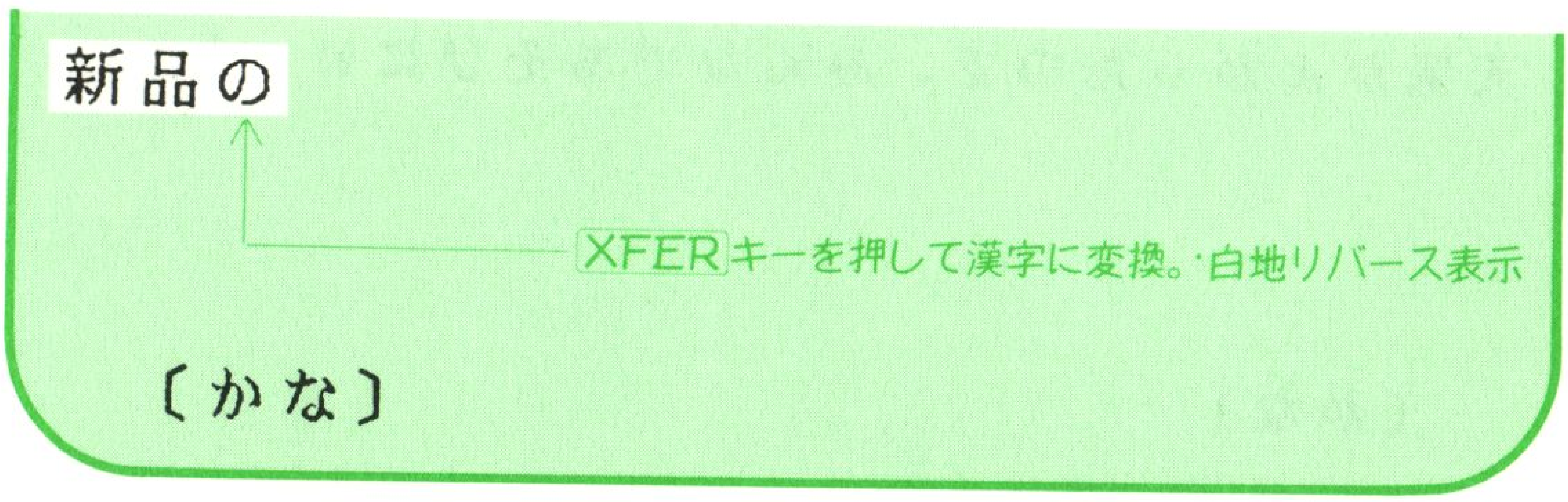
カタカナが入力できるのは、入力モードの表示が直接入力モードの［カナ］、あるいは間接入力モードの【カナ】のときです。

入力モードの表示を［カナ］あるいは【カナ】にするには、入力モードの表示が［かな］あるいは【かな】の状態のときに、ファンクションキーのf・3キーを押します。

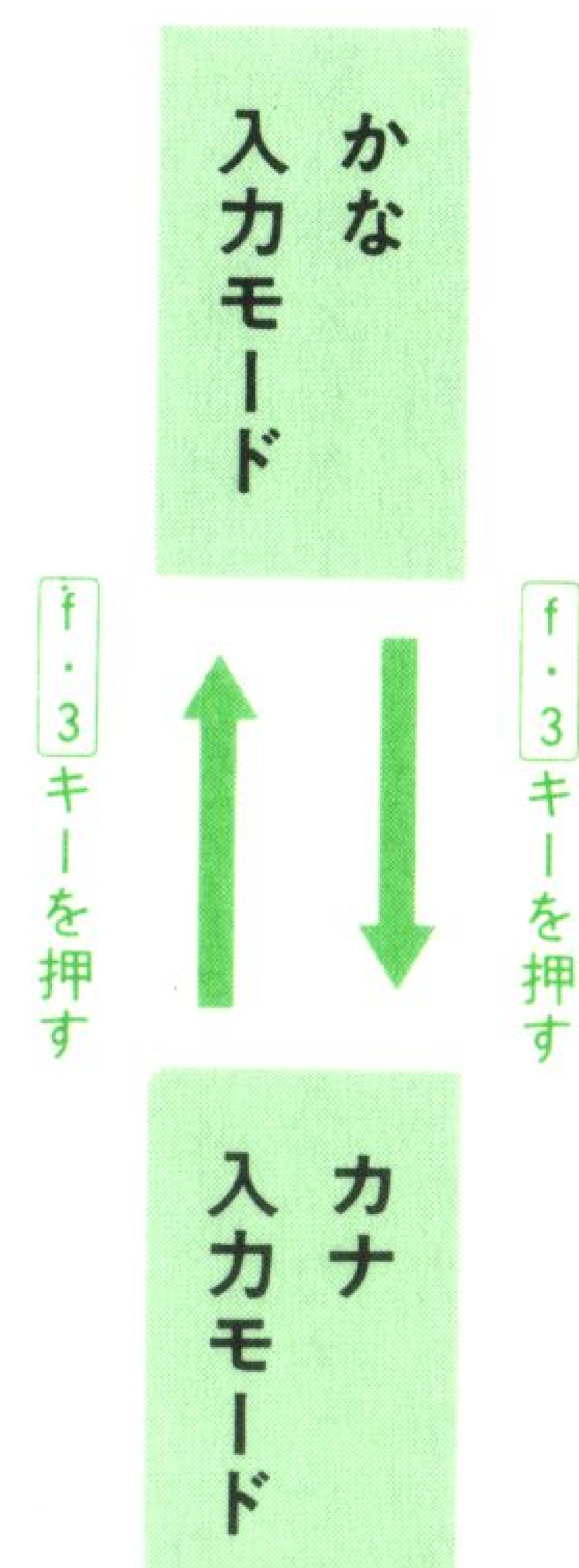
■F・3キーは「かな」と「カナ」の切り替え

f・3キーは、入力モードの「かな」と「カナ」を切り替えるキーで、「かな」から「カナ」に切り替えたあとf・3キーを押すと、「カナ」が「かな」に切り替わります。





カタカナの入力（1）



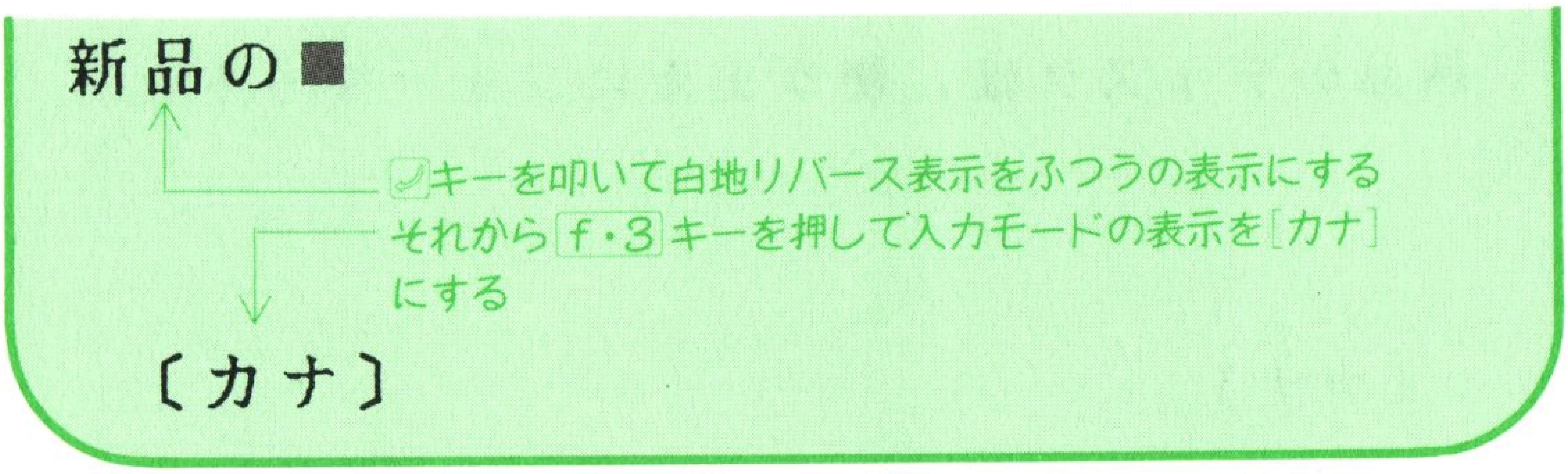
日本語入力モードでカタカナを入力

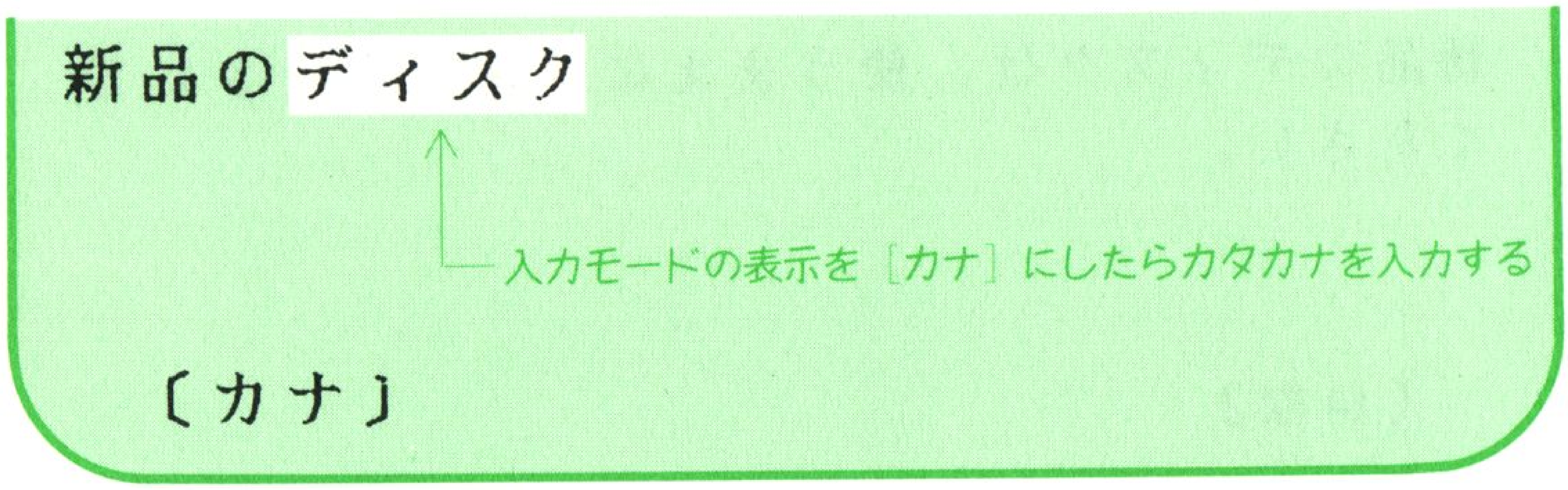
では実際に日本語入力モードで、カタカナを入力します。ここでは、つぎの文を直接入力モードで入力することにします。

新品のディスクは、使うまえにフォーマットしてください。

まず、入力モードが［かな］の状態で「しんぴんの」と入力し、XFERキーを押して「新品の」に変換します。変換したら、つぎは「ディスク」を入力します。

カタカナを入力する場合、f・3キーを押して入力モードを［かな］から［カナ］に切り替えなければなりませんが、漢字に変換した場合は「新品の」が白地リバース表示のままでは、f・3キーを押しても受けつけられません。





カタカナの入力（2）

そこで、[↵]キーを叩いて「新品の」をふつうの表示にし、それから[f・3]キーを押して、入力モードの表示を［カナ］に切り替えて「ディスク」と入力します。

「ディスク」のつぎは、「は、」の入力です。この場合は、「ディスク」が白地リバース表示のままでも、[f・3]キーを押して入力モードの表示を［カナ］から［かな］に切り替えることができますから、[f・3]キーで入力モードの表示を［かな］に切り替えて「は、」を入力します。

さて、つぎは「つかう」と入力して漢字に変換しますから、[↵]キーを叩いて白地リバース表示の「ディスクは、」をふつうの表示にし、「つかう」と入力して、[XFER]キーを押して「使う」に変換します。

この場合、「つかうまえに」までを入力して[XFER]キーを押しても、「使うまえに」と変換されませんから、注意してください。

「使う」を漢字に変換したら、「まえに」を入力します。すると「使う」はふつうの表示になり、「ま

新品のディスクは、使うまえにフォーマット

ひらがなが無変換入力の場合は、白地リバースの
ままでも[f・3]キーで入力モードの表示を[カナ]
に切り替えられる。
したがって、つづけて入力できる。

〔カナ〕

新品のディスクは、使うまえにフォーマットして
ください。

カタカナは、無変換入力のひらがなとおなじなの
で、[f・3]キーで入力モードの表示を［かな］に
切り替えることができる。
したがって、つづけて入力できる。

〔かな〕

カタカナの入力（3）

えに」が白地リバース表示になります。

ところで「まえに」は無変換入力ですから、「まえに」が白地リバース表示のままでも、[f・3]キーで入力モードの表示を切り替えることができます。したがって、[f・3]キーを押して入力モードの表示を［カナ］に切り替えて、「フォーマット」と入力します。

さて、「フォーマット」のあとに入力する「してください。」は漢字に変換する必要はありませんから、[f・3]キーを押して入力モードの表示を、「かな」に切り替えて入力します。

```
平均値を求めます。
終わりのときは、00を入力。

数を入れてください。?

平均値は　　です。
```

日本語でガイド文を画面に表示させる

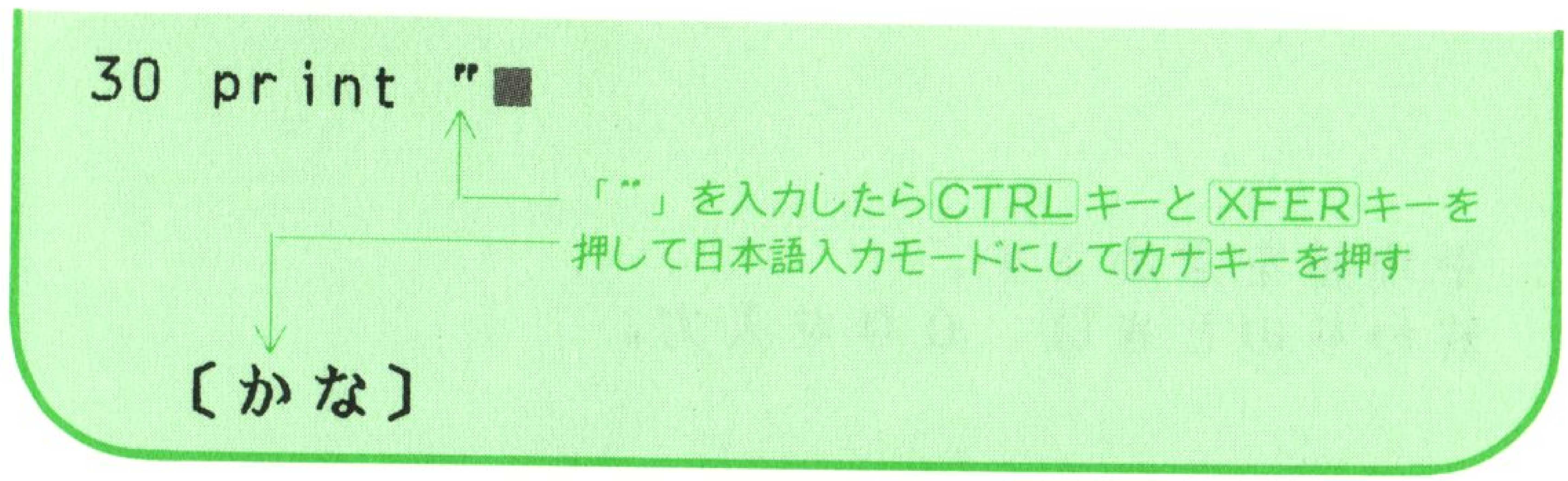
日本語で ガイド文を表示

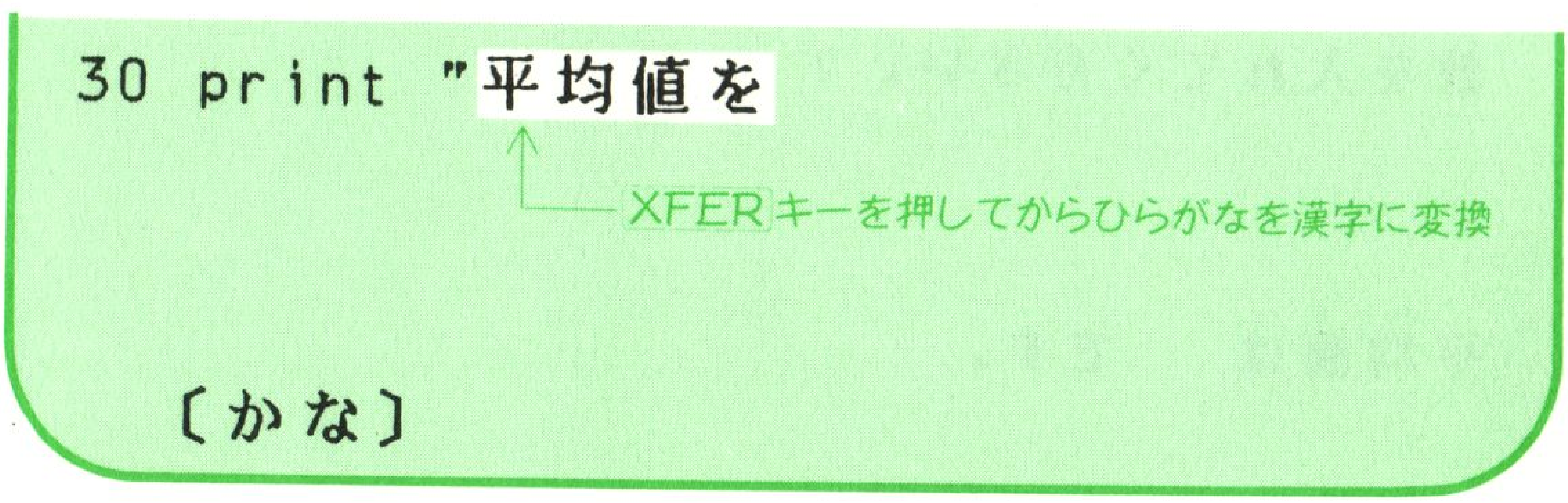
さきに日本語入力モードで、漢字、ひらがな、カタカナなど、日本語の入力の仕方について示しました。

プログラムのガイド文に日本語を使うと、プログラムを実行させたとき、画面に表示されるガイド文が非常にわかりやすくなりますから、それだけパソコンが使いやすくなります。

そこで、ここではまとめとして、平均値を求めるプログラムのガイド文に日本語を使って、上に示すように、画面に表示させることにします。平均値を求めるプログラムは、238頁に示します。

さて、PC－9801VF2、あるいはVM2に電源を入れ、VF2の場合はシステムディスク（2）を、VM2の場合はシステムディスクをド





ベーシックと日本語の入力（1）

ライブ1に挿入します。これでDisk versionが表示されて、ベーシックはもちろんのこと、日本語の入力ができます。また作成したプログラムをディスクにセーブすることもできます。

さて、238頁に示したプログラムは、行番号10のCLS文から入力していきますが、ここでは日本語がある行番号30と50のPRINT文と、行番号70のINPUT文の入力について取りあげます。

ベーシックと日本語の入力

まず、行番号の30を入力し、f・8キーを押してPRINTを入力し、つぎにSHIFTキーを押しながら、一番上の段にある2キーを押して、「"」引用符を入力します。

それから、CTRLキーとXFERキーを押して日本語入力モードにし、カナキーをロックします。これで入力モードの表示は、直接入力モードの「かな」になりますから、「へいきんちを」と入力してXFERキーを押して漢字に変換します。

30 print "平均値を求めます。

XFERキーを叩いて漢字に変換したら、⏎キーを押して白地リバース表示をふつうの表示にする

〔かな〕

30 print "平均値を求めます。"

ふつうの表示にしたらCTRLキーとXFERキーを押してベーシックの入力モードに戻す。カナキーのロックを解除して「"」を入力し、⏎キーを叩く

〔かな〕

ベーシックと日本語の入力（2）

つぎに「もとめます。」と入力し、XFERキーを押して漢字に変換します。

漢字に変換したら⏎キーを叩いて、白地リバース表示の「求めます。」をふつうの表示にし、そのあと、CTRLキーとXFERキーを押してベーシックの入力モードに戻します。そしてカナキーを押してキーのロックを解除して、「"」を入力し、⏎キーを叩きます。これで、行番号30のPRINT文の入力は終わりです。

つぎは行番号50のPRINT文です。行番号50のPRINT文の入力も、行番号30のPRINT文の入力とおなじです。ひらがなを、つぎのように入力してXFERキーを押すと、漢字に変換できます。

おわりのときは、＝終わりのときは、
00＝テンキーを叩いて入力　を＝無変換
にゅうりょく。＝入力。

この場合、「00を」は無変換入力です。00の入力は、入力モードの表示が［かな］の状態のままテンキーの0を叩いて入力し、そのあとつづけ

```
10 CLS ← 画面をクリアする
20 LOCATE 20,10 ← 表示位置を指定
30 PRINT "平均値を求めます。"
40 LOCATE 20,12 ← 表示位置を指定
50 PRINT "終わりのときは、00を入力。"
60 FOR T=1 TO 5000:NEXT:CLS ← ウエイト・ループ
70 INPUT "数を入れてください。";A ← データを入力
80 IF A=00 THEN 120 ← 00が入力されたら行番号120にジャンプ
90 B=B+A ← データの合計を求める
100 C=C+1 ← 何個のデータを合計したかカウントする
110 GOTO 70 ← 行番号70のINPUT文にジャンプ
120 HEIKINCHI=B/C ← 平均値を求め、変数HEIKINCHIに代入
130 PRINT "平均値は";HEIKINCHI;"です。"
                    ↑ 行番号120で求めた結果を表示
140 END
```

平均値を求めるプログラム

て「を」を入力します。それから↵キーを叩いて白地リバース表示の「00を」をふつうの表示にして、「にゅうりょく。」を入力します。

行番号70のINPUT文の入力も行番号30のPRINT文の入力とおなじです。

ひらがなを、つぎのように入力して[XFER]キーを押すと、漢字に変換されます。

かずを＝数を

いれてください。＝入れてください。

行番号130のPRINTの入力も、行番号30の入力とおなじですが、算出した平均値を記憶している変数HEIKINCHIを、ガイド文の間に置く関係から、ガイド文が二つにわけられています。「平均値は」を日本語入力モードで入力したら、ベーシックモードに戻して「";HEIKINCHI;"」を入力し、ふたたび日本語入力モードにして「です。」を入力します。

絵でわかるはじめてのPC-98

1989年8月15日　発　行　©

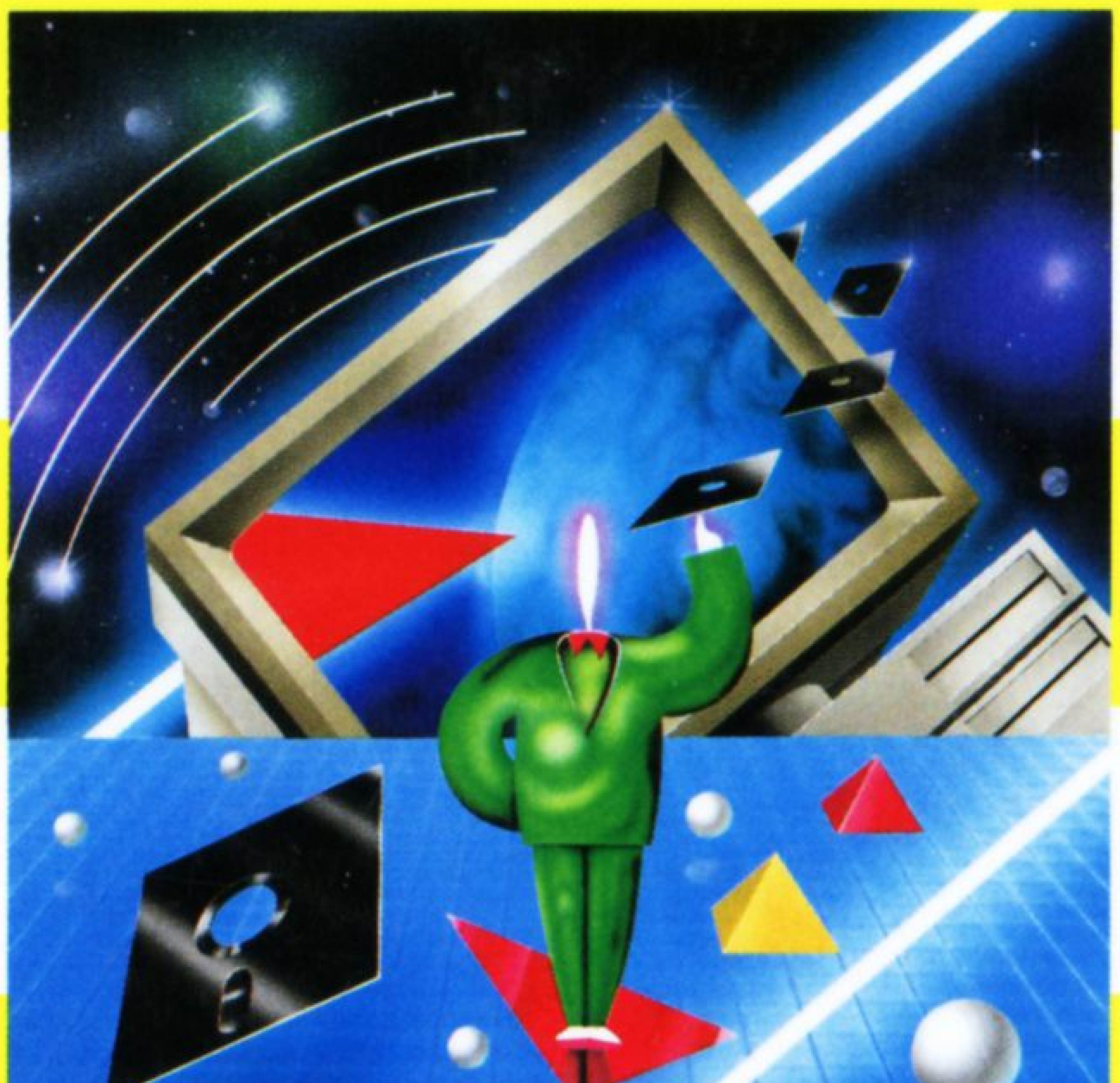
著　　者　　田　中　一　郎
発 行 者　　富　永　弘　一
印 刷 所　　湯島印刷株式会社

発行所　東京都台東区 台東4丁目7　株式会社 新星出版社
郵便番号110　電話(831)0743　振替東京 4-72233

ISBN4-405-06086-X

絵でわかる

はじめてのPC-98



新星出版社

ISBN4-405-06086-X C2055 P1000E

定価1000円
(本体971円)